

2

# NAVAL POSTGRADUATE SCHOOL

## Monterey, California

AD-A276 736



### THESIS

THE APPLICATION OF ARTIFICIAL-INTELLIGENCE  
TECHNIQUES TO THE AUTOMATIC IDENTIFICATION OF  
SOUNDS RECEIVED AT HYDROPHONES AND TO THE  
CORRELATION OF THESE SOUNDS BETWEEN  
HYDROPHONES.

by

Dennis A. Seem

December 1993

Thesis Advisor:

Dr. Neil C. Rowe

SDTIC  
ELECTE  
MAR 14 1994  
S E D

Approved for public release; distribution is unlimited.

94-08203



94 3 11 163

**Best  
Available  
Copy**

**REPORT DOCUMENTATION PAGE**Form Approved  
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503

<b>1. AGENCY USE ONLY (Leave Blank)</b>		<b>2. REPORT DATE</b> December 1993	<b>3. REPORT TYPE AND DATES COVERED</b> Master's Thesis, March 1992 - December 1993
<b>4. TITLE AND SUBTITLE</b> THE APPLICATION OF ARTIFICIAL-INTELLIGENCE TECHNIQUES TO THE AUTOMATIC IDENTIFICATION OF SOUNDS RECEIVED AT HYDROPHONES AND TO THE CORRELATION OF THESE SOUNDS BETWEEN HYDROPHONES. (U)			<b>5. FUNDING NUMBERS</b>
<b>6. AUTHOR(S)</b> Seem, Dennis Allen			
<b>7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)</b> Computer Science Department Naval Postgraduate School Monterey, CA 93943-5000			<b>8. PERFORMING ORGANIZATION REPORT NUMBER</b>
<b>9. SPONSORING/ MONITORING AGENCY NAME(S) AND ADDRESS(ES)</b> Naval Postgraduate School Monterey, CA 93943-500			<b>10. SPONSORING/ MONITORING AGENCY REPORT NUMBER</b>
<b>11. SUPPLEMENTARY NOTES</b> The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the United States Government.			
<b>12a. DISTRIBUTION / AVAILABILITY STATEMENT</b> Approved for public release; distribution is unlimited.			<b>12b. DISTRIBUTION CODE</b>
<b>13. ABSTRACT (Maximum 200 words)</b> The U. S. Navy's Integrated Underwater Surveillance System (IUSS) monitors hydrophones in the northeast Pacific. Geophysicists studying the IUSS data find seismic events and correlate them between hydrophones to locate their sources. Marine biologists and intelligence personnel are interested in the identification and localization of other sounds in the IUSS data. The current means of identifying and correlating sounds is a laborious visual examination of the data on a graphics workstation. In this thesis, a computer-vision method is presented for automatically identifying the sources of low-frequency sounds that are received on the IUSS hydrophones. Also presented in this thesis is a blackboard architecture for correlating sound "shapes" between hydrophones using a time-shift transform. The methods in this thesis properly identify approximately 90% of apparent whale moans and 100% of seismic events. The use of the time-shift transform has resulted in nearly a 100% success rate in correlating whale moans between far-field hydrophones, despite marked sound distortions with distance.			
<b>14. SUBJECT TERMS</b> Sound Correlation, Sound Identification, Artificial Intelligence, Blackboard Architecture			<b>15. NUMBER OF PAGES</b> 113
			<b>16. PRICE CODE</b>
<b>17. SECURITY CLASSIFICATION OF REPORT</b> Unclassified	<b>18. SECURITY CLASSIFICATION OF THIS PAGE</b> Unclassified	<b>19. SECURITY CLASSIFICATION OF ABSTRACT</b> Unclassified	<b>20. LIMITATION OF ABSTRACT</b> Unlimited

Approved for public release; distribution is unlimited

THE APPLICATION OF ARTIFICIAL-INTELLIGENCE TECHNIQUES TO THE  
AUTOMATIC IDENTIFICATION OF SOUNDS RECEIVED AT HYDROPHONES  
AND TO THE CORRELATION OF THESE SOUNDS BETWEEN HYDROPHONES

by

Dennis A. Seem  
Lieutenant Commander, NOAA Corps  
B.S., The George Washington University, 1976

Submitted in partial fulfillment of the  
requirements for the degree of

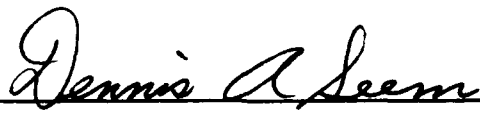
MASTER OF SCIENCE IN COMPUTER SCIENCE

from the


NAVAL POSTGRADUATE SCHOOL

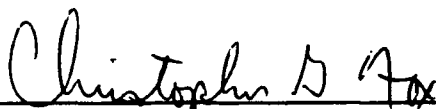
December 1993


Author:

  
Dennis A. Seem

Approved By:

  
Neil C. Rowe, Thesis Advisor

  
Christopher G. Fox, Second Reader

  
Ted Lewis, Chairman,  
Department of Computer Science

## ABSTRACT

The U. S. Navy's Integrated Underwater Surveillance System (IUSS) monitors hydrophones in the northeast Pacific. Geophysicists studying the IUSS data find seismic events and correlate them between hydrophones to locate their sources. Marine biologists and intelligence personnel are interested in the identification and localization of other sounds in the IUSS data. The current means of identifying and correlating sounds is a laborious visual examination of the data on a graphics workstation.

In this thesis, a computer-vision method is presented for automatically identifying the sources of low-frequency sounds that are received on the IUSS hydrophones. Also presented in this thesis is a blackboard architecture for correlating sound "shapes" between hydrophones using a time-shift transform. The methods in this thesis properly identify approximately 90% of apparent whale moans and 100% of seismic events. The use of the time-shift transform has resulted in nearly a 100% success rate in correlating whale moans between far-field hydrophones, despite marked sound distortions with distance.

Accession For	
NTIS	CRA&I <input checked="" type="checkbox"/>
DTIC	TAB <input checked="" type="checkbox"/>
Unannounced <input type="checkbox"/>	
Justification .....	
By .....	
Distribution /	
Availability Codes	
Dist	Avail and/or Special
A-1	

## TABLE OF CONTENTS

I.	INTRODUCTION.....	1
A.	GOALS.....	1
B.	ORGANIZATION.....	2
II.	SURVEY OF PREVIOUS WORK.....	3
A.	INTRODUCTION.....	3
B.	BACKGROUND.....	3
C.	T-PHASE PROCESSING.....	5
D.	COMPUTER VISION.....	6
E.	WHALE RESEARCH.....	7
III.	IDENTIFICATION OF DISCRETE SIGNALS.....	9
A.	INTRODUCTION.....	9
B.	AUTOMATIC IDENTIFICATION OF ACOUSTIC SIGNALS.....	9
C.	CHARACTERISTICS OF SIGNALS.....	11
IV.	CORRELATION OF SIGNALS.....	15
A.	INTRODUCTION.....	15
B.	NEAR-FIELD CORRELATIONS.....	16
C.	FAR-FIELD CORRELATIONS.....	18
V.	IDENTIFICATION PROGRAM.....	21
A.	INTRODUCTION.....	21
B.	GENERAL DESCRIPTION.....	21
C.	BLOCK DIAGRAMS.....	23
D.	LOW-LEVEL VISION PROCESSING COMPONENTS.....	26
E.	HIGH-LEVEL VISION PROCESSING COMPONENTS.....	31
F.	ABANDONED ALGORITHMS.....	33
VI.	CORRELATION PROGRAM.....	34
A.	INTRODUCTION.....	34
B.	GENERAL DESCRIPTION.....	34
C.	INPUT AND OUTPUT.....	35
D.	BLOCK DIAGRAMS.....	37
E.	DATA STRUCTURES.....	41
F.	PROGRAM COMPONENTS.....	43
G.	ABANDONED ALGORITHMS.....	45
VII.	DISCUSSION OF RESULTS.....	47
A.	INTRODUCTION.....	47
B.	IDENTIFICATION PROGRAM.....	47
C.	CORRELATION PROGRAM.....	49
VIII.	CONCLUSION.....	54
A.	INTRODUCTION.....	54
B.	ACHIEVEMENTS.....	54
C.	WEAKNESSES.....	55
	APPENDIX A: SAMPLE OF RESULTS FROM IDENTIFICATION PROGRAM.....	56

APPENDIX B: PLOTS PRODUCED BY THE CORRELATION PROGRAM .....	63
APPENDIX C: PROLOG CODE FOR THE IDENTIFICATION PROGRAM .....	70
APPENDIX D: CLOS CODE FOR THE CORRELATION PROGRAM .....	88
LIST OF REFERENCES .....	104
INITIAL DISTRIBUTION LIST .....	106

## LIST OF FIGURES

Figure 1:	Time-voltage and time-frequency plots of a signal. ....	10
Figure 2:	Time-frequency plot of a calibration signal.....	12
Figure 3:	Time-frequency plot of several whale moans.....	13
Figure 4:	Time-frequency plot of a t-phase signal. ....	13
Figure 5:	Time-frequency plot of a noisy signal. ....	14
Figure 6:	Top Level Diagram of Identification Program .....	23
Figure 7:	Diagram of Low-Level Vision Processing. ....	24
Figure 8:	Diagram of High-Level Vision Processing.....	25
Figure 9:	Results of the First Three Steps of Low-Level Processing.....	29
Figure 10:	Results of Low-Level Processing Steps Four To Seven.....	30
Figure 11:	Graphs Produced by the Correlation Program.....	36
Figure 12:	Diagram of Control Program. ....	37
Figure 13:	Diagram of Blackboard Controller. ....	38
Figure 14:	Diagram of Process New Region Program. ....	39
Figure 15:	Diagram of Match Far Regions Program.....	40
Figure 16:	Far-Field correlation of a Whale Moan. ....	50
Figure 17:	Maximum Far-Field Correlation Magnitudes.....	51
Figure 18:	Maximum Far-Field Correlation Magnitudes.....	52



# **I. INTRODUCTION**

## **A. GOALS**

There were two major goals of this thesis. The first goal was to develop a computer-vision method of automatically identifying the sources for discrete low-frequency underwater sounds. These sounds are received by fixed hydrophones located off of the west coast of the United States. The sounds are digitized at 128 Hertz. The second goal was to develop computer-vision techniques for automatically correlating the shapes of these signals to closely spaced pairs of hydrophones that are hundreds of kilometers apart.

Artificial-intelligence vision-processing techniques were used to achieve the first goal. Approximately 90% of large baleen whale moans are properly identified, nearly all t-phase signals from earthquakes are identified, and man-made calibration signals were also identified. The shape correlation goal was achieved through the use of an artificial-intelligence blackboard architecture and by computer-vision correlation of regions from two sets of distantly spaced hydrophones. A comparison of this method against the traditional correlation of signals using Fast Fourier Transforms shows this to be a valid technique.

There were two motivations for this thesis. Current processing on the acoustic signals attempts to automatically detect the occurrence of t-phase signals. There are many false identifications of t-phase signals that are actually whale moans. This thesis addresses that problem. The other motivation for this thesis is that the knowledge about blue and fin whale migration patterns and stock assessments is very limited. This thesis demonstrates that acoustic tracking of large baleen whales is feasible.

The techniques developed for this thesis can be used to build a system that will automatically track large baleen whales and to automatically detect and locate submarine seismic events.

## **B. ORGANIZATION**

Chapter II of this thesis discusses background about noises that are produced underwater and how these noises can propagate for long distances. Chapter II also reviews the previous work that has been done with t-phase detection and correlation, and with large baleen whale identification. Chapter III discusses the methods used to identify discrete low-frequency underwater signals. Chapter IV discusses the methods used to correlate signals between nearby hydrophones as well as far-apart hydrophones.

Chapter V gives a description of the computer program that was written to solve the identification problem. The identification program was written in the Prolog language. Chapter VI describes the computer program that was written to solve the correlation problem. The correlation program was written with the Common Lisp Object System (CLOS). Chapter VII is a discussion of the results of this research including the performance of the computer programs. Chapter VIII presents the conclusions of this thesis. Both achievements and weaknesses are discussed, as well as suggestions for further research.

## **II. SURVEY OF PREVIOUS WORK**

### **A. INTRODUCTION**

The United States Navy has recently declassified many aspects of its undersea acoustic surveillance systems (Fox, 1993). These systems are now being used by geophysicists in the VENTS program, the sponsor of this work, to monitor undersea earthquakes and volcanoes. The undersea surveillance system also has great potential for use in tracking and estimating populations of marine mammals, including the largest baleen whales. This chapter discusses background concerning the use of the undersea surveillance systems. An overview of the current state of t-phase processing is then presented. A brief description of computer vision is also presented along with its relevance to the processing of digitally recorded hydrophones. The chapter ends with a discussion of the research on large baleen whale acoustics.

### **B. BACKGROUND**

The National Oceanic and Atmospheric Administration (NOAA), Pacific Marine Environmental Laboratory (PMEL), Ocean Environment Research Division (OERD) is conducting an ongoing study of the impact that the dynamics of the geological processes occurring on the mid-ocean spreading ridges have on the Earth's oceans. Earthquakes and volcanism play a major part in these dynamic processes. This ongoing study is known as the VENTS program.

The United States Navy has for many years had an undersea surveillance system that has been used for monitoring submarine activities (Fox, 1993). This system consists of both fixed and mobile hydrophone systems. The system of interest in this thesis is the SOund SURveillance System (SOSUS). SOSUS consists of fixed arrays of hydrophones mounted at the approximate depth of the deep sound channel. These hydrophones are very sensitive and have the ability to pick up very low frequency sounds.

When an earthquake occurs, there are three kinds of energy, or phases, associated with it. First, there is a primary phase known as a p-phase. This phase consists of energy that is transmitted as compressional shock waves within the earth's crust. The secondary or s-phase consists of transverse shock waves which are also transmitted through the earth's crust. The tertiary phase of an earthquake, or t-phase, is only associated with underwater earthquakes. The t-phase is acoustic energy which is transmitted into the water column from the shaking of the ocean floor. The definition of t-phase has been expanded to include any low-frequency seismic event which transmits acoustic energy into the water column (Hammond, 1990). T-phase signals which are received at remote hydrophones typically have a frequency range from just a few Hertz to just a few tens of Hertz (Hammond, 1990).

There are two qualities of the physics of the ocean that allow us to study t-phases and other low-frequency acoustic phenomena at long ranges. One quality is that the attenuation of sound by seawater is very small for such low frequencies. The attenuation rate due to absorption in seawater varies with the square of the frequency. For a frequency of 100 Hz, the attenuation rate due to absorption is approximately  $10^{-6}$  decibels per meter (Clay, 1977, pg. 100). The second quality is the deep sound channel which exists in all of the world's oceans. This deep sound channel traps sound which enters it at a small enough angle and propagates it for very long ranges without the sound interacting with either the ocean floor or the ocean surface. This reduces the attenuation of sound due to spreading from a spherical relationship to a cylindrical relationship.

The speed of sound increases with increasing pressure and it decreases with decreasing temperature. A greater water depth correlates with a greater pressure at that depth. Also, a greater water depth generally correlates with a colder water temperature until the temperature reaches a few degrees above freezing. The temperature of the water in the ocean at depth does vary, but minimally. There are many components to the sound speed equation for seawater, including salinity, but the factors of pressure and temperature are primarily responsible for creating a sound speed minimum at a moderate depth in the ocean. The depth of the sound-speed minimum varies depending on the qualities of the water

column, but it is typically around 800 meters in mid-latitudes. Sound traveling in the water is refracted toward areas that have lower sound speeds. This sound speed minimum is the reason that the deep sound channel exists.

### **C. T-PHASE PROCESSING**

One of the goals of this thesis was to develop a technique which would automatically identify t-phases. Loud whale moans are sometimes mistakenly identified as t-phases and some low-volume t-phases are not detected. A discussion of the t-phase processing follows.

In 1991 the VENTS program installed a data collection system at a Navy facility to digitize and record data from the SOSUS hydrophones. NOAA/PMEL/OERD was given access to hydrophones from five hydrophone arrays scattered around the northeast Pacific ocean basin. Two hydrophones from each of five arrays, plus five directed beams, are being monitored continuously by the NOAA system. The hydrophone signals are digitized at a rate of 128 Hertz with 8 or 16 bit accuracy, depending on the hydrophone. The digitized hydrophone signals are recorded to 2.3-Gigabyte 8-mm tapes. Approximately one tape of data per week is collected (Fox, 1992).

The collected data is analyzed by scientists at OERD using signal processing methodologies. Fast Fourier Transforms are run on the digitized recordings from each hydrophone to transform the data from its time vs. amplitude raw format to a time vs. frequency format. This time-frequency data is scrolled across a computer screen while a scientist visually looks for t-phases. The analysis software also has an automatic t-phase detector built in, but it has not proved trustworthy enough to allow the data tapes to be scanned entirely by machine. After t-phase events are identified, they are correlated between hydrophones on the same hydrophone array. This correlation gives a time difference between nearby hydrophones. From this information, and with the information provided by using an underwater sound propagation model, a tentative hyperbolic line of position can be determined. The line of position is tentative because the correlations are

sometimes ambiguous. The best six correlations are saved in case the first correlation proves to be erroneous (Fox, 1992).

A rough location can be determined by intersecting the hyperbolic lines of position on a seismic event from three or more hydrophone arrays. The position obtained by using time differences of the closely spaced hydrophones is not very accurate because of the errors that are induced by the proximity of the hydrophones combined with the digitization rate of the signal. Once a rough position is obtained, a more accurate position can be determined by time-correlating the t-phase receptions between the widely spaced hydrophones of the different arrays (Fox, 1992).

#### **D. COMPUTER VISION**

For the t-phase processing, when the time-frequency data from the hydrophones is scrolled across the computer screen, the operator identifies the t-phases or other events of interest visually. T-phases have shapes which are easily distinguished by a human operator looking at the time-frequency plot. There are also whale moans in this data, and they have easily distinguishable shapes that are visually distinct from the t-phases. The signal processing techniques that are used in the analysis of these sounds are not designed to distinguish between the shapes of whale moans and the shapes of t-phases. The automatic detector that is in use is amplitude based, and it sometimes provides false detections due to whale moans or other noises (Fox, 1992).

Vision processing techniques are well suited for the t-phase problem. Time-frequency plots of the hydrophone data are digitized pictures. The t-phases in these plots are easily recognized by a human operator. They have simple parameters and can also be easily recognized by vision processing techniques. The whale moans are more complex, but they are also easily identified by a human operator. The parameters of whale moans are also more difficult to identify, but most whale moans can be recognized by vision processing techniques. A short discussion of computer vision processing follows.

Computer vision is the sub-field of artificial intelligence that is concerned with emulating human visual perception. The basic problem of computer vision is to identify the shapes that are in a digitized picture (Charniak, 1985, pg. 89). Vision processing is broken up into two distinct parts. Low-level processing is performed on the entire picture in order to gain information from the raw image and classify it for later processing. The results of this low-level processing is typically a set of edges, corners, and regions. The high-level processing uses the knowledge gained by the low-level processing to perform a cognitive analysis of the picture (Ballard, 1982, pg. 6). Usually, this high-level processing uses mathematical descriptions of typical shapes as its knowledge domain in order to identify the shapes in the picture (Charniak, 1985, pg. 150).

Low-level vision processing is centered around the concept of connected pixels. If a picture is made up of pixels that are either turned on or turned off, then strongly connected pixels are pixels that are turned on and are adjacent to each other, either vertically or horizontally. Weakly connected pixels are pixels that are turned on and have an adjacent corner. Disconnected pixels are those pixels that are turned on and are neither strongly nor weakly connected (Dougherty, 1992).

This thesis directly uses some vision processing programs by Professor Neil C. Rowe of the Naval Postgraduate School. These programs perform the low-level vision processing needed to create a black-and-white image from a multi-spectral image and they also identify regions for further processing.

## **E. WHALE RESEARCH**

During the course of the research for a thesis on t-phase detection and correlation, it became apparent to us that there might be some benefit from this thesis to the community of whale researchers. There are many sounds on the data tapes that we examined that are from some large species of whale. If the species of whale can be determined, and if an average rate of moaning can be determined, then the techniques presented in this thesis can

be used to help determine stock assessments, locations, and migration patterns of these whales.

The whale moans that occur repeatedly in this data are not a known call from any species of whale (McDonald, 1993). There has been speculation that these moans come from either blue whales (*Balaenoptera musculus*) or fin whales (*Balaenoptera physalus*) (Fox, 1993b). These two species of whale are the largest living species of animals (Mizroch, 1984a; Mizroch, 1984b), and both species have been previously associated with low-frequency moans, although these moans have had different shape and duration (Bird, 1987; Cummings 1971; Cummings, 1986). The inability of researchers to associate the moans that are seen in the t-phase data with a particular species indicates the inherently difficult nature of studying large whales that live in the open ocean.

Both fin whales and blue whales are either solitary or are found in small groups and both species also migrate north-south over vast stretches of ocean each year (Mizroch, 1984a; Mizroch, 1984b). The stock assessment of each of these species of whale is very rough due to the difficulties of counting solitary animals in the open ocean (Mizroch, 1984a; Mizroch, 1984b). As of 1984, the stock estimates of the blue whale in the north Pacific indicate that it has not made any recovery from the time that the killing of blue whales by man ceased in 1965 (Mizroch, 1984a).



### **III. IDENTIFICATION OF DISCRETE SIGNALS**

#### **A. INTRODUCTION**

The objective of the signal identification part of this thesis was to identify whale moans and t-phase signals. Complicating this objective is the environment within which the hydrophones exist. On the data segments that we analyzed, there were some time periods that were extremely noisy, and also some time periods where the hydrophone signal was replaced with a calibration signal. There was some noise present for the entire data set. This chapter discusses the methodology used to identify regions from a noisy data set.

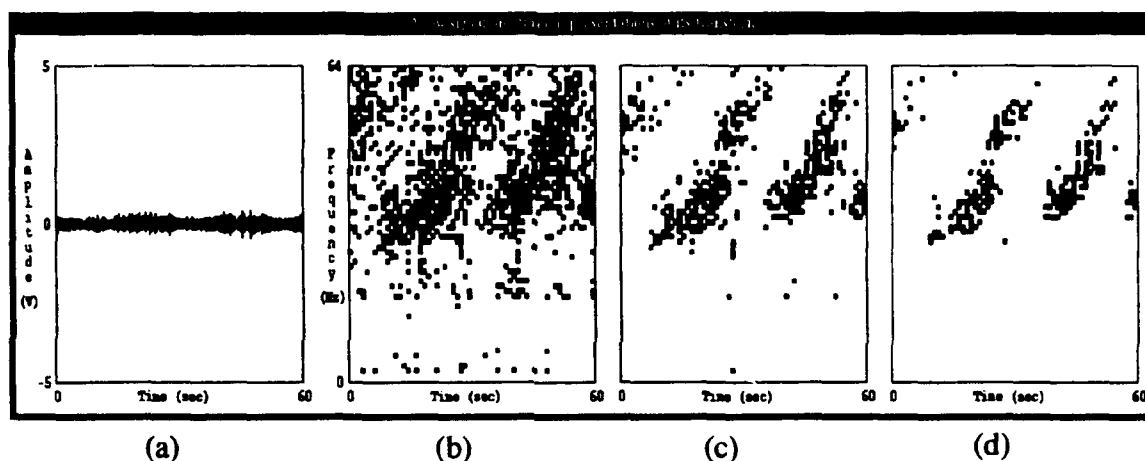
#### **B. AUTOMATIC IDENTIFICATION OF ACOUSTIC SIGNALS**

Previous efforts to automatically identify t-phase signals have been unreliable because they have been based on the amplitude of the incoming signal in a selected frequency band. Anything that makes a loud enough noise in the same band will be detected as a t-phase. Also, any earthquake that does not make a loud enough noise will not be detected as a t-phase. An amplitude detector is the best method to use with traditional signal processing methods, even with the limitations, due to the nature of these methods.

We have taken a different approach to automatically identifying t-phase signals. This approach also has the side-benefit of identifying most other signals that are received on the hydrophone. By looking at the time-frequency plot of the hydrophone data as a picture, we use computer-vision processing methods to identify the signals that are digitized from the hydrophone. This emulates the process of a human operator looking at the data and identifying signals of interest.

Figure 1 shows a time-voltage plot of some sounds received by a hydrophone and the transform of that signal into the time-frequency domain. Figure 1(a) is a plot of the raw signal voltages over time. There is one minute of signal reception plotted, or 7680 points of raw data. Figure 1 (b), (c), and (d) show this same signal after it has been Fourier transformed into the time-frequency domain. Transforms were performed on each second

of data, giving a one second resolution in the time scale and a one Hertz resolution in the frequency scale. Only one side of the symmetric spectrum from 0 to 64 Hertz are shown. The black pixels in Figure 1(b) are all of the pixels in the Fourier transform that had a magnitude greater than 15. In Figure 1(c), the black pixels are those with a magnitude greater than 30. In Figure 1(d), the black pixels are those with a magnitude greater than 45. Each element of a Fourier transform array represents a magnitude of sound over a range of frequencies. For these Fourier transform arrays, each element of each Fourier transform array represents the magnitude of the sound intensity received for one second over a range of one Hertz.



**Figure 1: Time-voltage and time-frequency plots of a signal.**

The geophysicists that are monitoring the hydrophone signals for t-phases are not burdened with the problem of representing magnitudes in black and white. The processing is being done on color graphics workstations. Magnitudes are represented by an array of colors. The analyst working with the data can immediately see the intensity of the sound over various frequencies by the color-coded output.

T-phases, whale moans, and other identifiable signals have distinct shapes that can be recognized by both a human and a computer. In order for a computer to recognize a signal, the signal must first go through some low-level vision-processing which will take the time-frequency data and find cohesive regions while eliminating a majority of the noise. We

have developed a ten-step low-level processing strategy that takes a set of Fast Fourier Transform magnitudes, arranged by time and frequency, and creates a set of well-defined regions that can be identified by number.

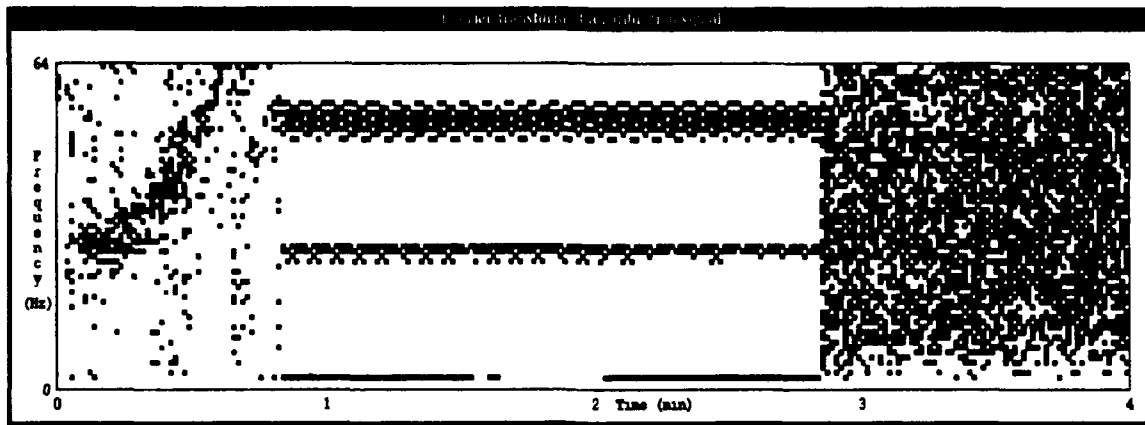
Our low-level vision processing creates, in essence, a black and white (or on and off) pixel image from the Fast Fourier Transform magnitudes. Noise is removed from this image and then regions have gaps filled in. Man-made pixels are then marked and removed. Man-made regions come from such things as boat noises and calibration signals. Once all of this is done, regions are clumped together. The final low-level step is to associate the turned-on pixels with the magnitudes that existed in the original data. Chapter V, Section D provides details on these steps.

Once the low-level processing is completed, high-level vision processing performs the feature extraction from the identified regions. This high-level processing groups nearby regions together into larger logical regions and determines whether they are t-phase signals, whale moans, or of unknown origin. This is done in a twelve-step process which is described in detail in Chapter V, Section E.

### **C. CHARACTERISTICS OF SIGNALS**

The calibration signal, which is removed in the low-level processing, always has the pixels at 27 and 28 Hertz turned on. It also always has the pixels between the frequencies of 3 and 24 Hertz turned off and the pixels between the frequencies of 30 and 34 Hertz turned off. Other pixels may or may not be turned on, but this description is adequate for uniquely describing the calibration signal.

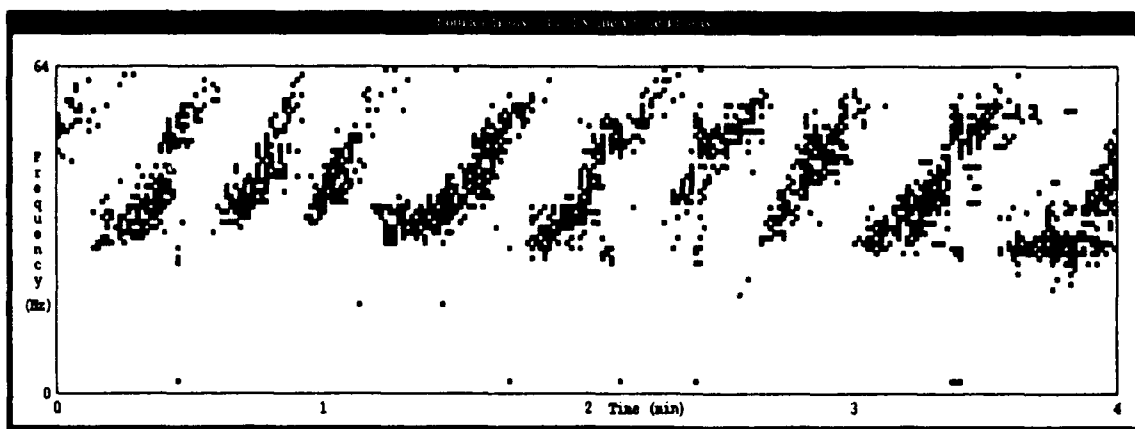
Figure 2 shows a typical calibration signal. In the first minute of this plot, there is a whale signal and some random noise. The calibration signal runs for approximately two minutes. During that two minutes, the reception of the hydrophone signal is cut off. After the calibration signal stops at about the third minute in the plot, the hydrophone signal becomes extremely noisy. For this figure all Fourier transform magnitudes that were greater than 45 were plotted.



**Figure 2: Time-frequency plot of a calibration signal.**

Through an examination of the Fourier transformed sounds, we found that whale moans meet the following minimum criteria: a minimum frequency greater than 20 Hz, a time span between 12 and 60 seconds, an area between 50 and 500 pixels, and a density between 8 and 75%. This very general set of criteria has been determined to be good enough to identify a majority of the whale signals without being so general as to mistakenly identify other signals as whales.

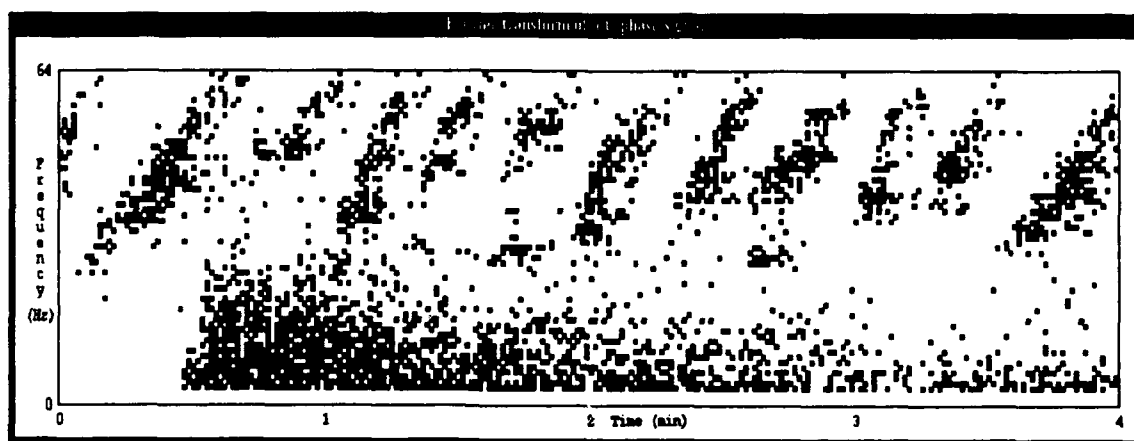
Figure 3 shows eight complete whale moans in the center of the time-frequency graphs and parts of two more whale moans at the beginning and ending times in the graph. The shapes are easily identifiable because human visual processing can fill in disconnected regions. Each of the whale moans has several properties in common, but each moan is also a distinctly different sound. For this figure, all Fourier transform magnitudes that were greater than 35 were plotted.



**Figure 3: Time-frequency plot of several whale moans.**

Every t-phase signal that we have observed has a minimum frequency of less than 10 Hertz. Many t-phase signals have the additional characteristic that their maximum frequency was less than 20 Hertz. These were subdivided into a class called far-field t-phases since the higher frequencies were probably frequency attenuated.

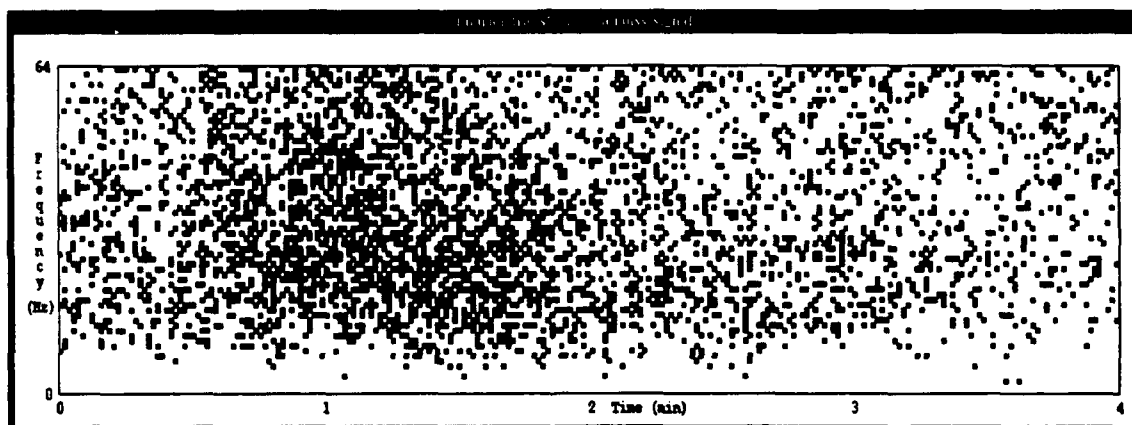
Figure 4 shows a t-phase signal in addition to whale moans. The t-phase begins at about the half minute point in the plot and slowly dies out by the fourth minute. There are also 10 complete and two partial whale moans during this time. For this figure, all Fourier transform magnitudes that were greater than 35 were plotted.



**Figure 4: Time-frequency plot of a t-phase signal.**

All pixel regions which did not meet any of these criteria, were classified as unidentified signals. Some of these unidentified signals were actually whale moans, but many of them were just unidentifiable noisy signals.

Figure 5 shows a particularly noisy time period. There are no discernible shapes in this signal. The cause of this noise could possibly be a defective link between the hydrophone and the data collection equipment. We speculate that this is the cause because the calibration signal shows up frequently around the noisy areas of the signal. Extremely noisy periods also occur when there is vessel traffic very near the hydrophones. For this figure, only Fourier transform magnitudes that were greater than 130 were plotted. Nearly every pixel in this graph has a magnitude greater than 45.



**Figure 5: Time-frequency plot of a noisy signal.**

## IV. CORRELATION OF SIGNALS

### A. INTRODUCTION

When an underwater sound is correlated between two hydrophones, the relative time-shift between the receptions of that sound can be determined. Once the time difference of a signal reception is known, a hyperbolic line-of-position can be found. Deterministic models exist for calculating lines-of-position, given the transmission qualities of the medium, the geometry of the hydrophone placements, and the time difference of the signal reception (Loje, 1983).

Four hydrophone recordings were studied for this thesis. The hydrophones are in a geometry of two pairs of hydrophones that are far apart. The hydrophones in each pair are approximately 400 meters apart and the pairs of hydrophones are approximately 340 kilometers apart (Dziak, 1993). Assuming a sound speed of 1500 meters per second in the deep sound channel, the travel time for sound between the nearby hydrophones is about a third of a second, and the travel time for sound between the far-apart hydrophones is nearly 4 minutes.

The objective of the signal correlation part of this thesis was to correlate identified sounds between the nearby hydrophones and then to correlate these nearby-correlated signals between the far-apart pairs of hydrophones. There are two different problems presented when attempting to correlate a signal between the nearby hydrophones and between the far-apart hydrophones. When correlating a signal between nearby hydrophones the time difference is short, but the differences in hydrophone characteristics, the near-field effects of the environment, and the effects of region growing in the signal identification program will make the signals look different. When correlating a signal between far apart hydrophones, attenuation and time stretching will have additional effects on the signal shape (Urick, 1975). There are also many more possible signals from which to choose for a correlation.

## **B. NEAR-FIELD CORRELATIONS**

The geophysicists that are monitoring the SOSUS hydrophones find the time-differences between the arrivals of signals at nearby hydrophones by using Fast Fourier Transform correlation techniques. The purpose of this nearby hydrophone sound correlation is to determine tentative a tentative bearing to the source of a signal. The preliminary bearings are calculated in order to reduce the number of calculations that are needed to correlate signals to the far-apart hydrophones. "The results are often ambiguous and a selection of six possible azimuths (and back-azimuths) are recorded" (Fox, 1992).

The purpose of correlating signals to the nearby hydrophones in this thesis was to determine possible candidates for correlation to the far-apart hydrophones. Finding tentative bearings from the time offsets was not possible using our approach because of the resolution of the shapes created in the identification program was too rough relative to the travel time of sound between the nearby hydrophones. The research in this part of the thesis concentrated primarily on the whale moans. The whale moans are much more frequent than the t-phase signals and thus present a better opportunity to apply artificial-intelligence techniques to their correlation.

An artificial-intelligence computing model known as the blackboard architecture was applied to the problem of correlating signals to nearby hydrophones. The blackboard architecture was appropriate for this application because there is a hierarchy of states to which the regions can belong, and there are independent knowledge sources that can change the state of the blackboard based on the facts which have been posted to the blackboard.

The framework for a blackboard consists of three elements: the blackboard, multiple knowledge sources, and a controller. The blackboard consists of facts that are arranged in a hierarchical structure. The knowledge sources act independently on the facts contained in the blackboard and can add new facts and retract old facts. The controller determines when a knowledge source should be invoked (Nii, 1986).



The structure of the blackboard for matching the nearby hydrophones consisted of: unmatched regions, tentatively matched regions, definitely matched regions, and unmatchable regions. The characteristics of each hydrophone were also a part of the blackboard structure, along with the times associated with the matched regions.

The knowledge sources for the correlation program were: current-time, time-span, time-separation, best-match, unmatchable-signal, and multiple-match. Each of these knowledge sources contributed information to the blackboard, and when new knowledge was added to the blackboard, the knowledge sources were able to act on it. Some knowledge sources are more complex than others. The current-time knowledge source just adds the start time of the most recently read region to the blackboard while the unmatchable-signal knowledge source tests the current time against the start times of all of the regions in the unmatched-region list and moves regions to the unmatchable-region list based on the maximum possible travel time between the hydrophones.

The controller got a new region for a hydrophone and implemented the appropriate knowledge sources. The new region would either be added to the unmatched-region list or the tentatively-matched-region list. Once this was done, other knowledge sources would be called on to move unmatched regions to the unmatchable-region list and to move tentatively-matched regions to the definite-matched-region list.

Signals were matched by time shifting one signal so that its starting time matched the start time of the other signal. Then the number of pixels that the regions had in common was found. Doubling the number of overlapping pixels and dividing this by the sum of the pixels in both regions produced a number between 0 and 1 that represented a goodness-of-fit for the two signals. Each signal was time-shifted five times to cover a time period of two seconds on either side of the starting times of the signals. The maximum goodness-of-fit was taken from each of these time shifts. Multiple time shifts were needed because distortions of these signals during their transformations into shapes.

### C. FAR-FIELD CORRELATIONS

The geophysicists that are monitoring the SOSUS hydrophones find tentative bearings from the nearby hydrophones and use these bearings to search the data of the far-apart hydrophones for possible correlations of events. This can sometimes result in signals being incorrectly matched and it can also result in signals not being matched at all when they should be matched.

We took a different approach in this thesis. We narrowed down the total number of regions that could be matched by the approximate direct-line travel time of sound from one hydrophone pair to the other. In this case, the total travel time was approximately 250 seconds which includes an error factor for not knowing the transmission characteristics of the water column. For any particular sound received on one hydrophone pair, without a-priori knowledge about which direction that sound was coming from, there is a window of approximately 500 seconds of sounds to check against from the other hydrophone pair. For the whale moans in the data set that we examined, there were usually 15 to 20 moans that needed to be tested from one hydrophone pair for each moan recorded on the other pair.

Even though time stretching and attenuation affect a signal traveling over such a long distance, we found that using the same time-shift and overlap technique that was applied to the correlation of signals to the nearby hydrophones was sufficient to match signals between far-apart hydrophones. The region from each nearby hydrophone pair that had the largest area was used for the correlation. We chose to use the region with the largest area for matching because it gave better results than using the smallest region. The larger regions tended to have higher correlations with regions from the far-apart hydrophones.

Each pair of regions from the nearby hydrophones were matched to all possible pairs of regions from the far-apart hydrophones. The matchings were put into a list in decreasing order according to their goodness-of-fit. A time-shift transform was then performed on the best three matches for all of the regions.

We developed the time-shift transform from an inspiration given to us by the Hough transform. The Hough transform is a vision processing technique which is typically used

for finding lines and curves in an image. One of its best advantages is that gaps in lines and a noisy image have little effect on the results of the transform (Ballard, 1982, p. 123). We were not looking for lines, but trends in the time delays that were exhibited by the matches of the signals. Finding trends in the time delays, in an image where there are several very close goodness-of-fit values for time delays, is analogous to finding lines in a noisy image. We divided up the total possible time-delay for correlation of signals between the far-apart hydrophones into bins that covered approximately 25 seconds each. This gave us 20 bins for the time-delay that we worked with on this data set. We chose this number because 25 seconds was a typical length for a whale moan. For each region, the goodness-of-fit value for each of the top three matching regions was added to the appropriate time-delay bin. Each bin was then normalized to 0 by subtracting the smallest bin value from all of the bins. An omnidirectional noise level would show up as the same value in each of the bins in the time shift array. The normalization to 0 effectively eliminated these noise values.

After normalization to 0, each bin was replaced with its square root. The square root was applied because each bin is made up of the goodness-of-fit values from both sets of hydrophones. A large set of strongly correlated signals would overshadow a smaller set of correlated signals because of the contributions of both sets of hydrophones to the time-shift array. If each bin were divided by two, there would be no net overall effect, but by taking the square root, a dampening effect is applied to the strongly correlated values.

After each bin was replaced by its square root, each bin was normalized to 1 by dividing it by the maximum bin value. The result was a time-shift array with values in the range from 0 to 1. This time-shift array represented the relative frequencies of time delays for the correlations.

Once the time-shift array was determined, the goodness-of-fit value for each match between the far-apart hydrophones was multiplied by the appropriate value from the time-shift array. This application of the time-shift array to the best-fit values decreased the likelihood that a goodness-of-fit value, for a time delay that had little evidence to support it, would dominate a similar goodness-of-fit value, for a time delay that had a lot of

evidence to support it. All far-apart matches were then rearranged according to the new set of goodness-of-fit values.

After the application of the time-shift array, each pair of regions from one pair of nearby hydrophones had its best fit matched against the best-fit regions from the far-apart pair of hydrophones. The best fit of a pair of nearby hydrophones was a pair of nearby regions from the far-apart hydrophones. If the best fit of the far-apart pair of regions was the original set of regions, then those two sets of regions were classified as being correlated. After finding the best fits, each of the matched regions was removed from the list of possible matches and a second set of best fits was correlated. Two passes of the data set were sufficient to correlate nearly all of the far-apart regions.

## **V. IDENTIFICATION PROGRAM**

### **A. INTRODUCTION**

The identification program identifies whale moans, t-phase signals, and calibration signals. This chapter describes the identification program in detail, including the program input and output, program structure, and detailed descriptions of each program component.

### **B. GENERAL DESCRIPTION**

The identification program was written in Prolog. Prolog is a language that is extensively used in artificial-intelligence applications. Some attractive features of Prolog are its logic-like syntax, its ability to backtrack through previous steps, and its multidirectional reasoning (Rowe, 1988, p. xiv). It was chosen as the language for the first part of this thesis primarily for these reasons. It was also chosen because Professor Rowe had a library of computer-vision software, written in Prolog, that could be used as a foundation for the thesis research. One of the characteristics of Prolog programs, including the program written for this thesis, is that there is no main program as in a traditional programming language. The program is made up of many short programs, usually just a few lines long.

The identification program, diagramed in Figure 6, is approximately 1150 lines long, including the Prolog code and comments. I wrote approximately 750 lines and Professor Rowe wrote the other 450 lines. There is an additional library of vision processing programs that were written entirely by Professor Rowe. Some parts of this library are called on extensively by the programs in this thesis. This library contains approximately 750 lines of Prolog code.

The identification program (see Appendix C) starts with a listing of the Fast Fourier Transform (FFT) magnitudes which have been computed from the raw time-voltage data using a C program. The FFT magnitudes are in a time-frequency order. Each line contains

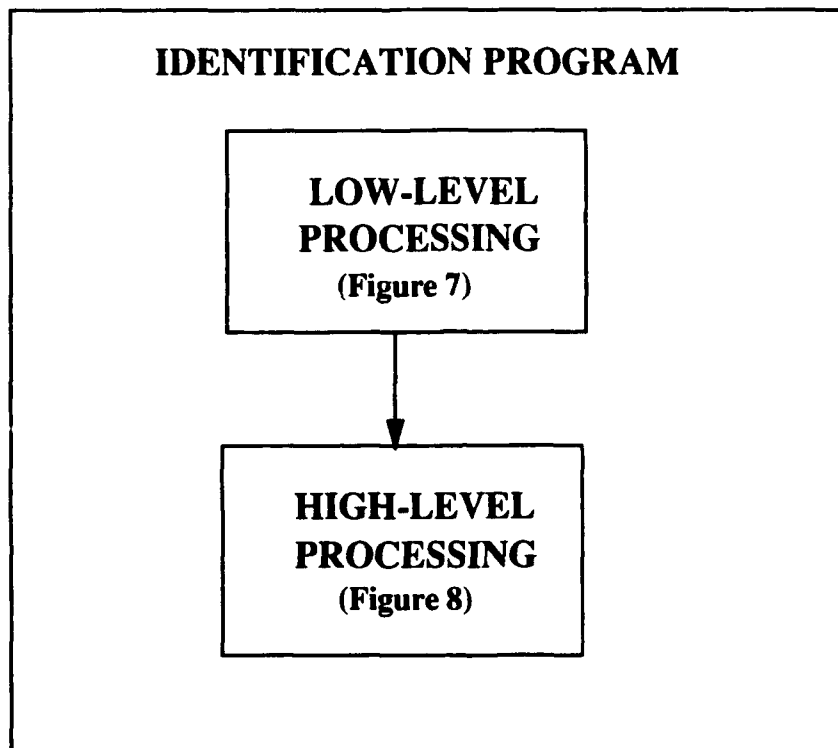
the magnitudes for one second of time and each of the 64 magnitudes on a line correspond to one Hertz of frequency in ascending order.

The primary purpose of the identification program is to take the FFT magnitudes, create a digital black-and-white picture from them, and identify the different shapes that appear in the picture. The program goes through two phases of computer-vision processing in order to achieve a result. The low-level processing works with the pixels of the entire picture to develop regions. The high-level processing performs feature extraction using the individual regions and classifies regions and groups of regions.

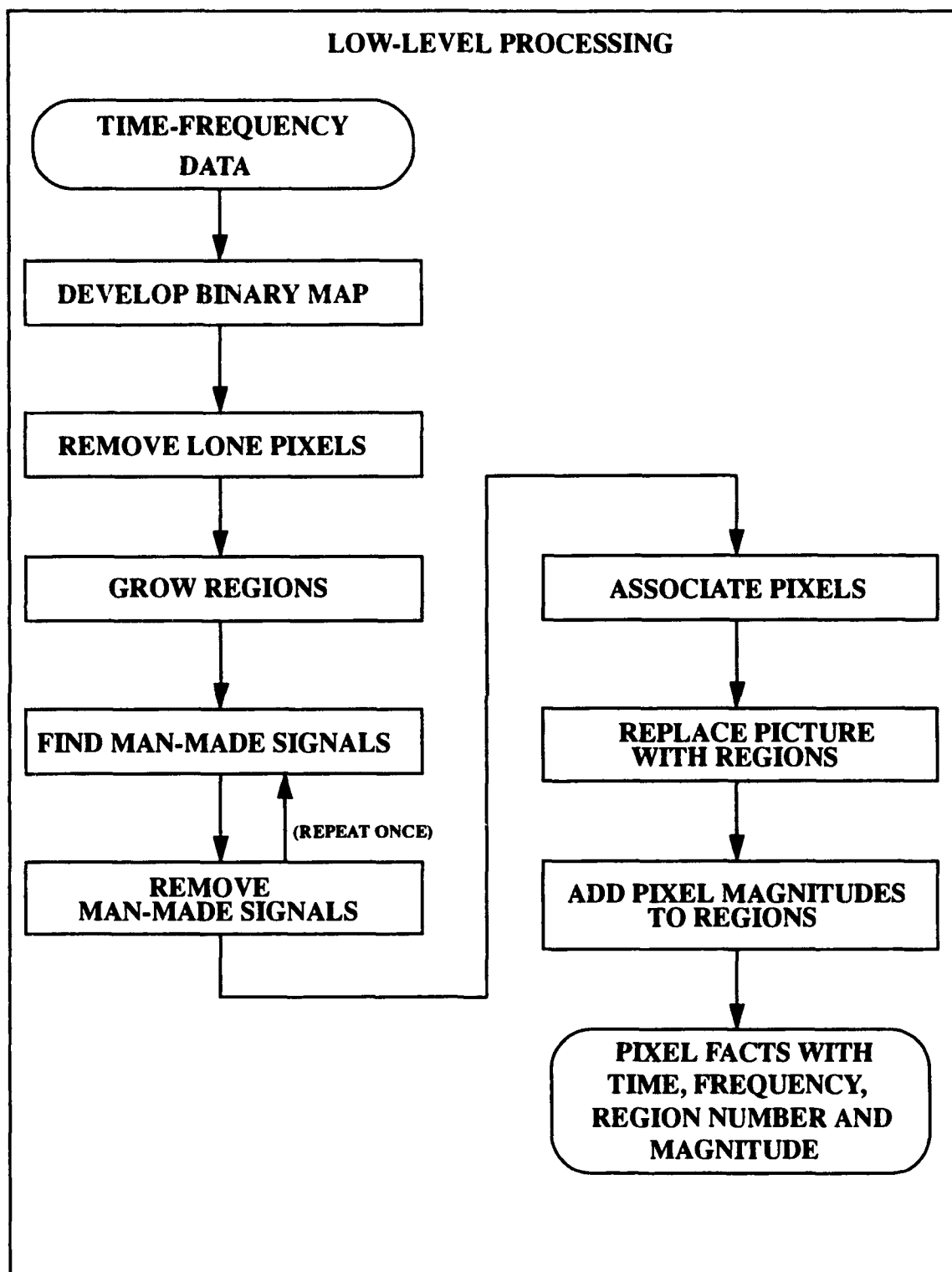
There are two sets of output from the identification program. The first set of output is a file which contains a list of the grouped regions, the parameters of these groups, and horizontal and vertical lines that have been found in the picture. The first set of output also contains a listing of the classifications of all of the groups of regions. These classifications are: whale, near-field t-phase, far-field t-phase, and unknown. The parameters of the groups are: minimum and maximum frequency, start and stop time, area, density, and average intensity. These parameters were chosen because they provide a good overall description of the region. The average intensity is defined as the sum of all of the magnitudes of the Fourier transform for each pixel in the group divided by the area of the group. The density is the pixel density in the bounding box delineated by the minimums and maximums of frequency and time. In addition to the previous items, the first set of output also contains an orientation for each unknown group of regions. A straight line is fit to the pixels in each unknown group in a least-squares manner and the orientation of that line is listed.

The second set of output is a list of pixels from each of the identified groups. This second set of output is written in such a manner as to be easily readable by the correlation program. Each group of regions is a list enclosed in parentheses with a letter code indicating the identification of the region as the first element of that list.

### C. BLOCK DIAGRAMS



**Figure 6: Top Level Diagram of Identification Program**



**Figure 7: Diagram of Low-Level Vision Processing.**



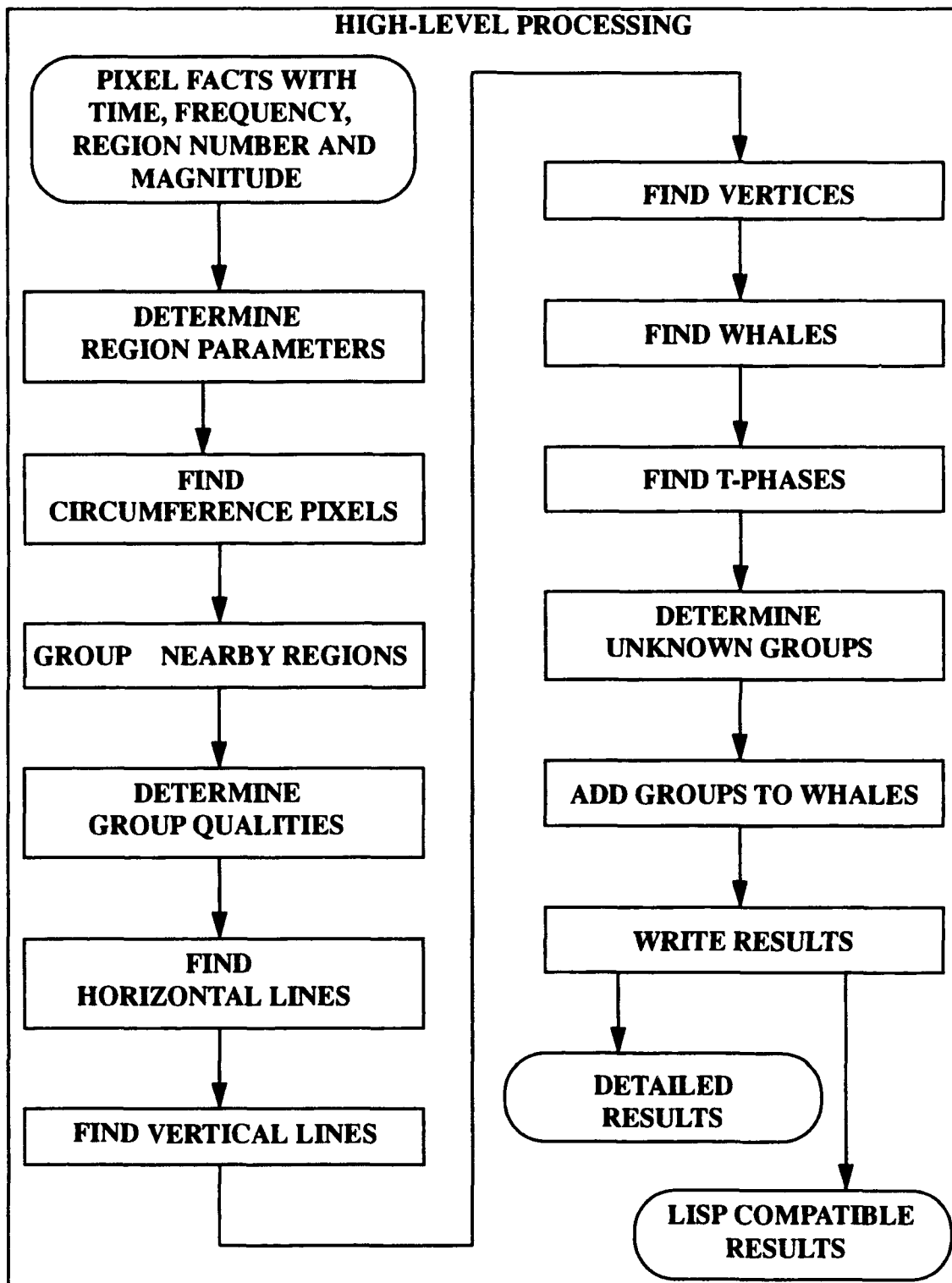


Figure 8: Diagram of High-Level Vision Processing.

#### **D. LOW-LEVEL VISION PROCESSING COMPONENTS**

The low-level processing is the first major section of the identification program. It consists of 10 steps which will take as input the time-frequency data and turn it into a set of identified cohesive regions.

When a Fast Fourier Transform (FFT) is run on the original time-amplitude data, it produces a set of data that is three dimensional in nature. For each time and frequency point, there is a number that represents the strength of the signal at that point. For the data set that was provided for this project, the original data was collected at 128 Hz. FFTs were run on every 128 points to get a 1 second time resolution and a 1 Hz frequency resolution. The FFT returns a frequency array of 128 points, but only 64 points are saved due to the symmetry of the spectrum (Press, 1988, pg. 403).

The first step in the low-level processing takes the time-frequency data and develops a binary map of pixel data that is based on a moving threshold value. The objective of the first step is to develop a black and white image that has enough pixels turned on so that shapes can be determined, but not so many pixels turned on so that multiple shapes become merged. Frequency values that were above the threshold level caused the corresponding pixel to be turned on and frequency values that were below the threshold level caused the corresponding pixel to be turned off. We experimentally determined that the threshold should be adjusted to keep the number of turned-on pixels between 15% and 25% of the number of pixels in the frequency range for one second. These values allowed the following steps to create a set of cohesive identifiable regions. Upper and lower limits were set on the moving threshold value so that extremely noisy periods and extremely quiet periods would be represented. Figure 9(a) shows some whale moans that are the result of the first step. The programs for creating a binary map of pixel data based on a fixed threshold were written by Professor Rowe and modified by the author for the moving threshold value.

The second step in the low-level processing takes the black and white pixel image that is the result of step one and turns off all pixels that are turned on and are not strongly connected. This step removes much of the spurious noise in the image. Figure 9(b) shows

the result of the second step. The program for performing the second step was written by Professor Rowe. The third step in the low-level processing performs region growing by turning on pixels in order to fill in gaps across rows, columns, and diagonals. Figure 9(c) shows the result of the third step. The effect of this region growing step is to make fuller shapes.

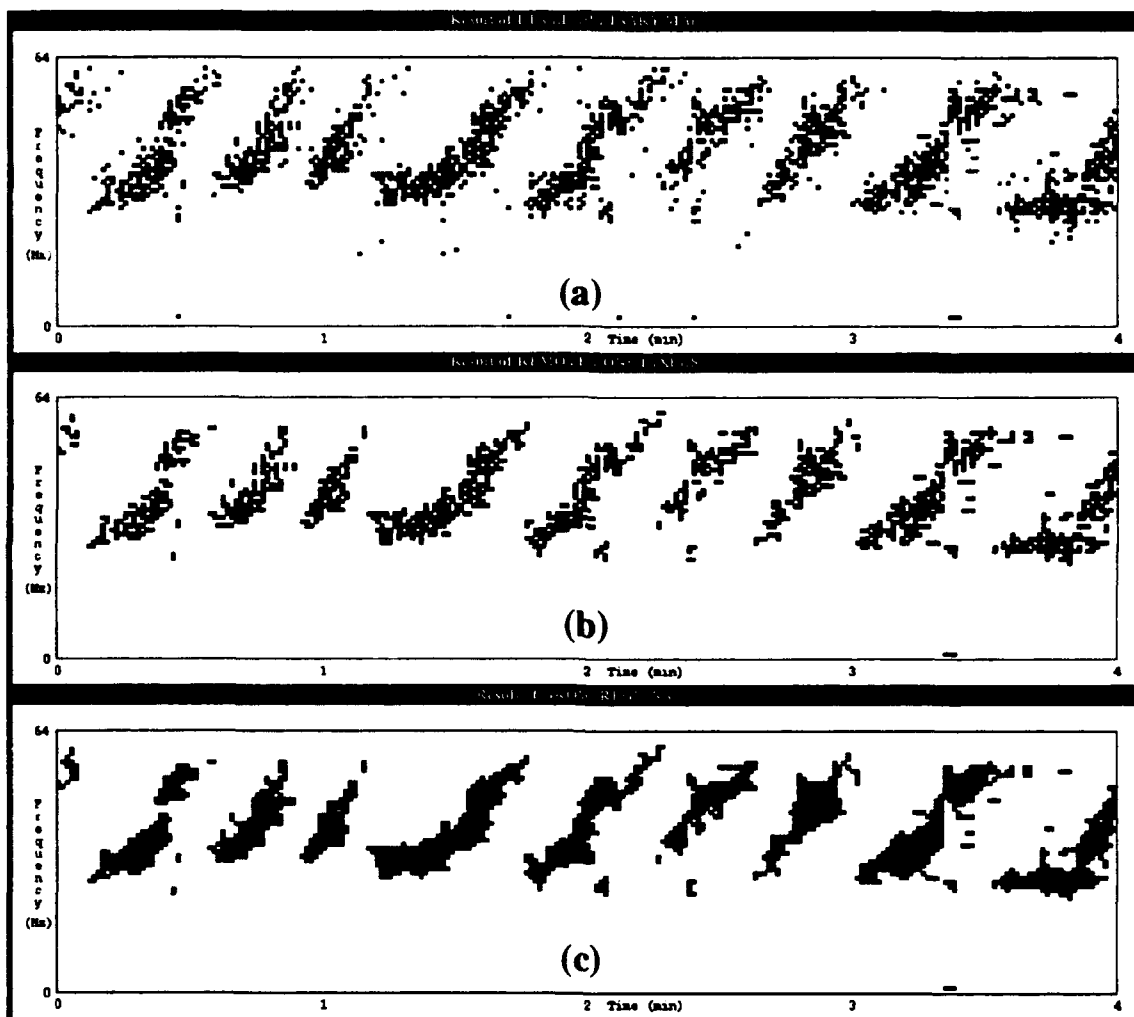
The fourth step and sixth step are the same, as are the fifth and seventh steps in the low-level processing. In the fourth and sixth steps, pixels which are probably noise or are man-made are marked with special symbols so that they can be removed in the next step. Presumed man-made pixels are those turned-on pixels that create a narrow frequency band for 2 or more seconds. Also fitting this category is the calibration signal, which has a distinctive set of frequencies. Pixels that are considered to be noise are those turned-on pixels that are not strongly connected in frequency, regardless as to whether they are strongly connected in time. The fifth and seventh steps in the low level processing simply remove all of the specially marked symbols from the previous steps. The result of these four steps is a black and white picture that has its major features sharpened and most of the noise surrounding these features removed. Figure 10 shows the results of these steps. The diagonal lines with a positive slope indicate pixels that were identified as man-made and the diagonal lines with a negative slope indicate pixels that were identified as calibration signals.

The eighth step in the low-level processing performs region clumping by taking the black and white picture and associating a region number with each pixel. All pixels that are either weakly connected or strongly connected are associated with the same region. The result of this clumping is a picture that has numbers in the place of all of the previously turned on pixels. The programs for performing the eighth step were written by Professor Rowe.

The ninth step in the low-level processing removes the concept of a picture that has pixels that are turned on or off and replaces this with the concept of a set of regions that have time and frequency values associated with them. The programs for performing the

ninth step were written by Professor Rowe. The tenth, and final, step in the low-level processing goes back to the original data file and associates the strength of the signal at each turned-on pixel with the time-frequency values for each region.

There is no longer an image for the vision processing methods to work with. Instead, there is a set of well defined regions that have time, frequency, and strength values associated with them. In low-level vision processing, each element of a picture is processed in the same way (Charniak, 1985, pg. 95). Even though some of the follow-on processing will process each element of each region in the same way, there aren't any turned-off pixels to process, so all of the following processing is considered to be high-level vision processing.



**Figure 9: Results of the First Three Steps of Low-Level Processing.**

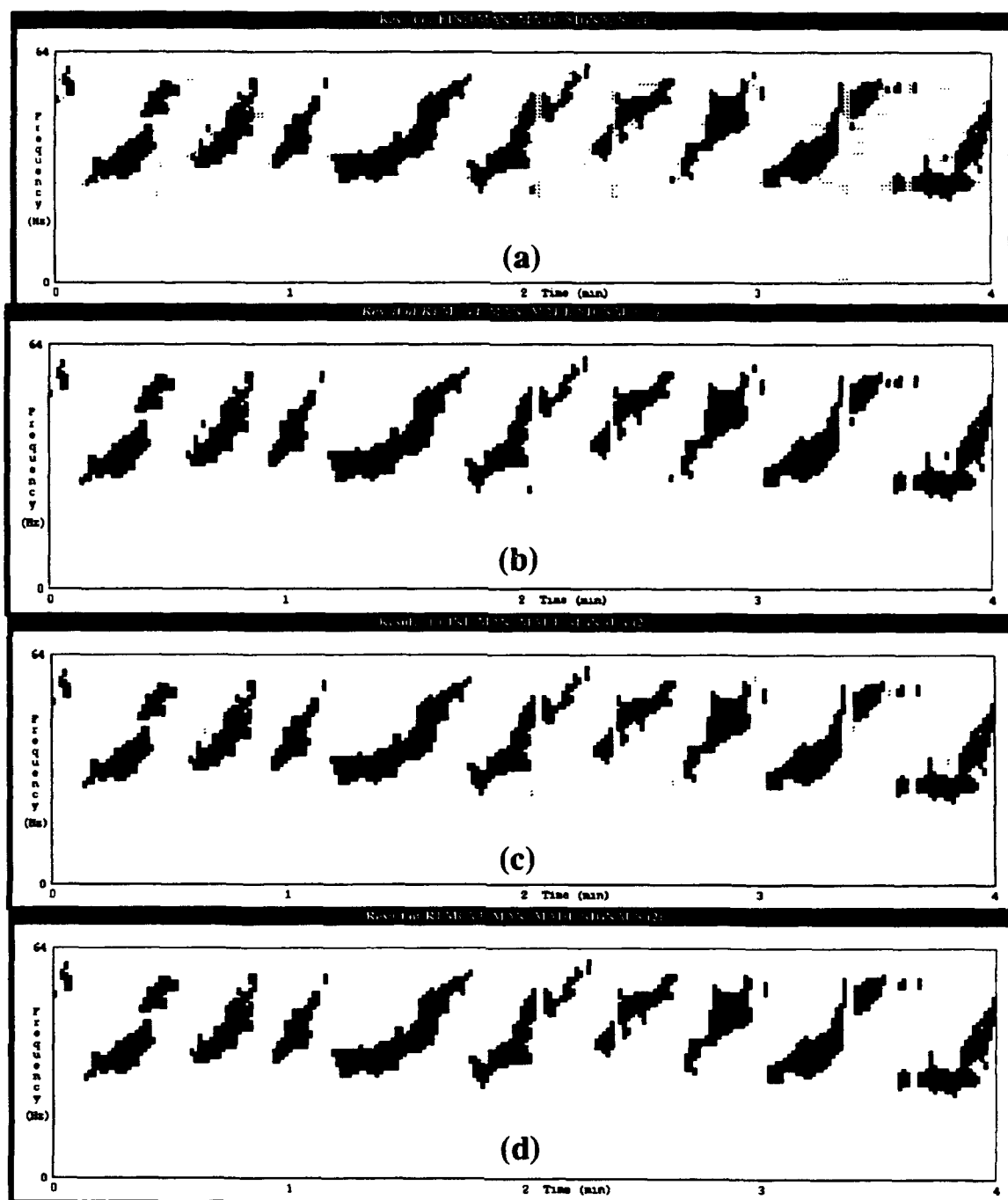


Figure 10: Results of Low-Level Processing Steps Four To Seven

## **E. HIGH-LEVEL VISION PROCESSING COMPONENTS**

The first step in the high-level processing finds the area, circumference, length, and width of each region. The area of a region is the total number of pixels in that region. The circumference of a region is the number of pixels in that region that are not strongly connected to four other pixels. The length of a region corresponds to the total amount of time that a sound occurs, and the width of a region corresponds to the total bandwidth of a signal.

The second step in the high-level processing is a computational convenience. It was found that by specially noting the circumference pixels, that the process of grouping regions was much faster. This step finds each pixel in each region that is a circumference pixel and asserts it as an outer pixel.

The third step finds nearby regions and groups them. If the square of the distance between any two outer pixels of two regions is less than 13, then the two regions were placed in the same group. The distance between pixels is related to their time-frequency value. The number 13 was derived experimentally.

The fourth step characterizes the groups. For each group, the start and stop times are determined and the minimum and maximum frequencies are determined. Each group also has its total area (total number of pixels) determined along with its density and average intensity. Density was defined as total area, which is determined by the bounding box, divided by the group area, which is a count of the number of pixels in the group. The average intensity was determined from the intensity values that were associated with each pixel in the last step of the lower-level processing.

The fifth step of the high-level processing finds horizontal lines. Horizontal lines are a collection of pixels that are outer-pixels of regions and extend over a significant period of time without changing in frequency. These lines could indicate the presence of a man-made signal that was not removed in the low-level processing. The sixth step is similar to the fifth step except that it finds vertical lines. These are lines that cover many frequencies at one time. The vertical lines could also be an indicator for a man-made signal such as an

underwater explosion. The seventh step in the high-level processing finds vertices from the intersections of the vertical and horizontal lines. These three steps are most often performed as low-level vision processing functions (Ballard, 1982, pg. 119), but we felt that it was more appropriate to perform these steps in the high-level processing so that region identifications could be associated with these features.

The eighth step is a first pass at finding whale signals. The criteria for a group to be called a whale on the first pass is that it has: a minimum frequency greater than 20 Hz, a time span between 12 and 60 seconds, an area between 50 and 500 pixels, and a density between 8 and 75%.

The ninth step in the high-level processing is finding t-phase signals. First, all potential t-phases are found with the simple criteria that they have an area greater than 5 pixels and a minimum frequency that is less than 10 Hz. T-phases are then divided into far-field and near-field earthquakes. Far-field earthquakes are considered to be those that have a maximum frequency of less than 25 Hz. See Chapter II for a discussion of frequency attenuation. Far-field earthquakes also have the tendency to have many disconnected groups of regions. If any two groups of regions are determined to be far-field earthquakes and they are separated by a time period of less than 9 seconds, then they are determined to be a part of the same earthquake. The near-field earthquakes are all earthquakes that are not far-field earthquakes.

The tenth step in the high-level processing is to place all of the remaining groups of regions that have an area greater than seven into a category called "unknown groups". For each of these unknown groups, further determinations are made about their shapes. A straight line is least-squares fit through the pixels of the group and the angle of that line in the frequency-time plane is determined. With this line established, the mean distance of the pixels from the line is determined, along with the variance.

The eleventh step in the high-level processing is to check each unknown group to see if it might be part of a whale moan. The previous processing was not including many of the upper frequency sounds of the whale moans because the regions of the higher frequency



sounds were separated by too many turned-off pixels from the regions making up the whale-moan group. If the maximum frequency of a whale moan is within 8 Hertz of the minimum frequency of an unknown group and if the ending time of a whale moan is within 14 seconds of the starting time of an unknown group, then the unknown group is removed from the list of unknown groups and a combined whale group is created.

The twelfth and final step in the high-level vision processing is writing the results to files. An explanation of the output is given in the general program description section of this chapter.

## **F. ABANDONED ALGORITHMS**

The whale moans could be classified using other types of parameters. Another parameter that can help to define a whale moan is the typical angle that a line would make if it were passed through the signal using a least squares fit. This would describe the tendency of the whale moan to upsweep in time through a frequency band of about 40 Hertz. This line-fitting approach was tried at one point in the investigation, but it was found to be too time consuming.

The calibration signal was originally left for high-level processing. It was found that frequent small gaps in the signal and variations in the upper frequency ranges of the signal, combined with abutting regions from other sources at the beginning and end of the calibration signal, caused it to be very difficult to identify during the high level processing, but the low-level processing identifies the calibration signal very easily on a line by line basis.

## **VI. CORRELATION PROGRAM**

### **A. INTRODUCTION**

The correlation program has two goals. The first is to correlate signals between nearby hydrophones. The second is to correlate the nearby-correlated signals to far-apart hydrophones. Different techniques were used for each type of correlation. This chapter describes the correlation program in detail, including the program input and output, program structure, data structures, and detailed descriptions of each program component.

### **B. GENERAL DESCRIPTION**

The correlation program (see Appendix D) was written in the Common Lisp Object System (CLOS). CLOS is an object-oriented programming language which is built on top of Common Lisp. It is used extensively in artificial-intelligence applications (Koschmann, 1990, pg. 5). We chose CLOS for this part of the thesis because of its ability to handle complicated data structures and because of the graphics capabilities that came with the implementation that was supplied with our computer systems. Our implementation was Allegro Common Lisp from Franz.

There are approximately 1000 lines of code in the program divided up into five files that are separated by function. Comments are sparse in the code because descriptive names were used for the variables, making comments redundant. It is possible to have global variables in a CLOS program, but it is not considered good programming practice, so there are none in this program.

The correlation program (see Appendix D) starts by creating and initializing instances of a blackboard data structure for each pair of nearby hydrophones. Then it creates and initializes a data structure for the graphic output. The blackboard control program is then called sequentially for each nearby pair of hydrophones. After each set of nearby hydrophones is processed, regions are correlated between the far-apart hydrophones.

### C. INPUT AND OUTPUT

The input to the program consists of lists of regions with the identification of the region type as the first element of the list. This is the output from the identification program. There are four hydrophones to which sounds are correlated. The identification program only processes the sounds from one hydrophone, so it must be run four times covering the same time period during each run, in order to produce the files that are used by the correlation program.

The output of the correlation program is an X-windows plot of identified and correlated regions in a time-frequency diagram (Figure 11(a) and (b)). The vertical axis is frequency from 0 to 64 Hertz. There are four hydrophones shown in the plot, so there are four vertical axes, all with the same scale. The nearby pairs of hydrophones are shown adjacent to each other on the plot, with the line between each pair corresponding to 64 Hertz on the lower hydrophone plot and 0 Hertz on the upper hydrophone plot.

The horizontal axis is time. There is approximately 20 minutes of sounds in each plot. Each minute from the start of the plot is denoted by a tic mark below the lower hydrophone in each pair. All hydrophones were processed over the same time period so there are no time offsets of the regions between hydrophones.

Regions are identified by a letter slightly above and on the right side of the region. Whales are identified with a "W", t-phases with a "Q", unknowns with a "U", and others with an "O". Correlated regions have a line drawn between them. The endpoints of the lines were determined by the middle pixel in the list of pixels for the region, so the angle of the line, especially between the nearby hydrophones does not represent the exact time offset for the correlation.

The numbers above the upper pair of hydrophones and below the lower pair of hydrophones are the time offsets for the near-field correlations. They are in whole seconds and are referenced to the left side of the graph.

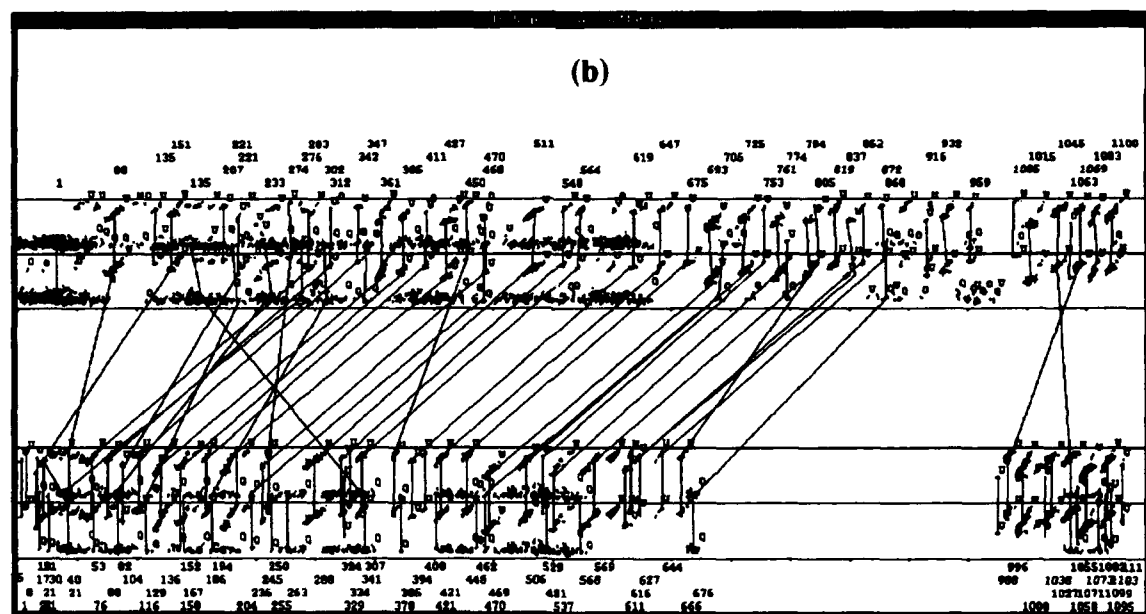
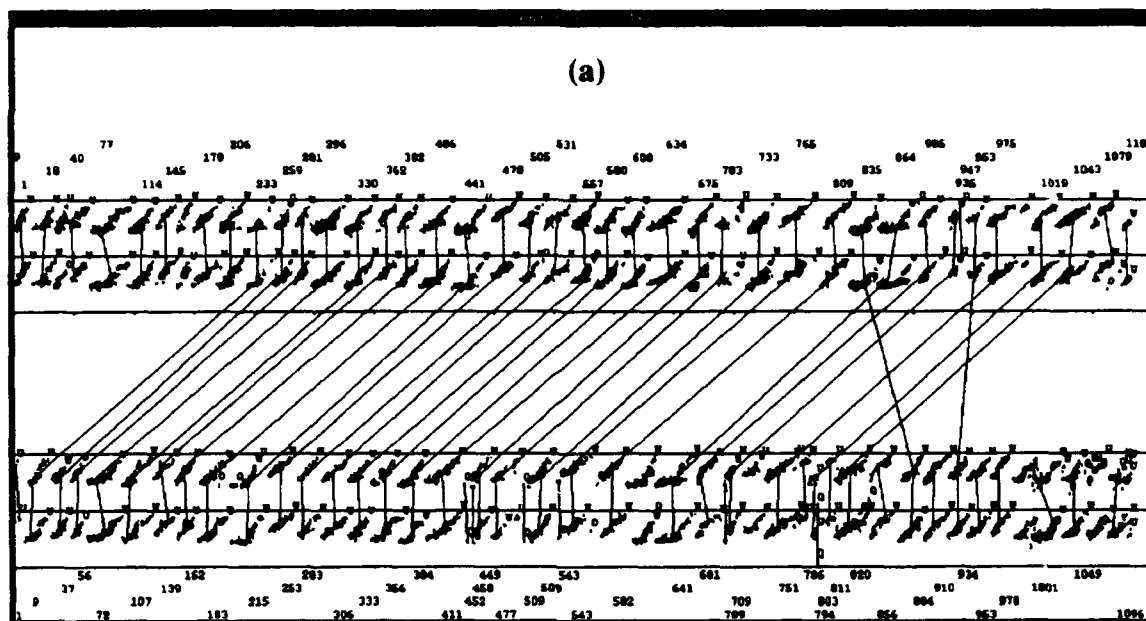


Figure 11: Graphs Produced by the Correlation Program.

#### D. BLOCK DIAGRAMS.

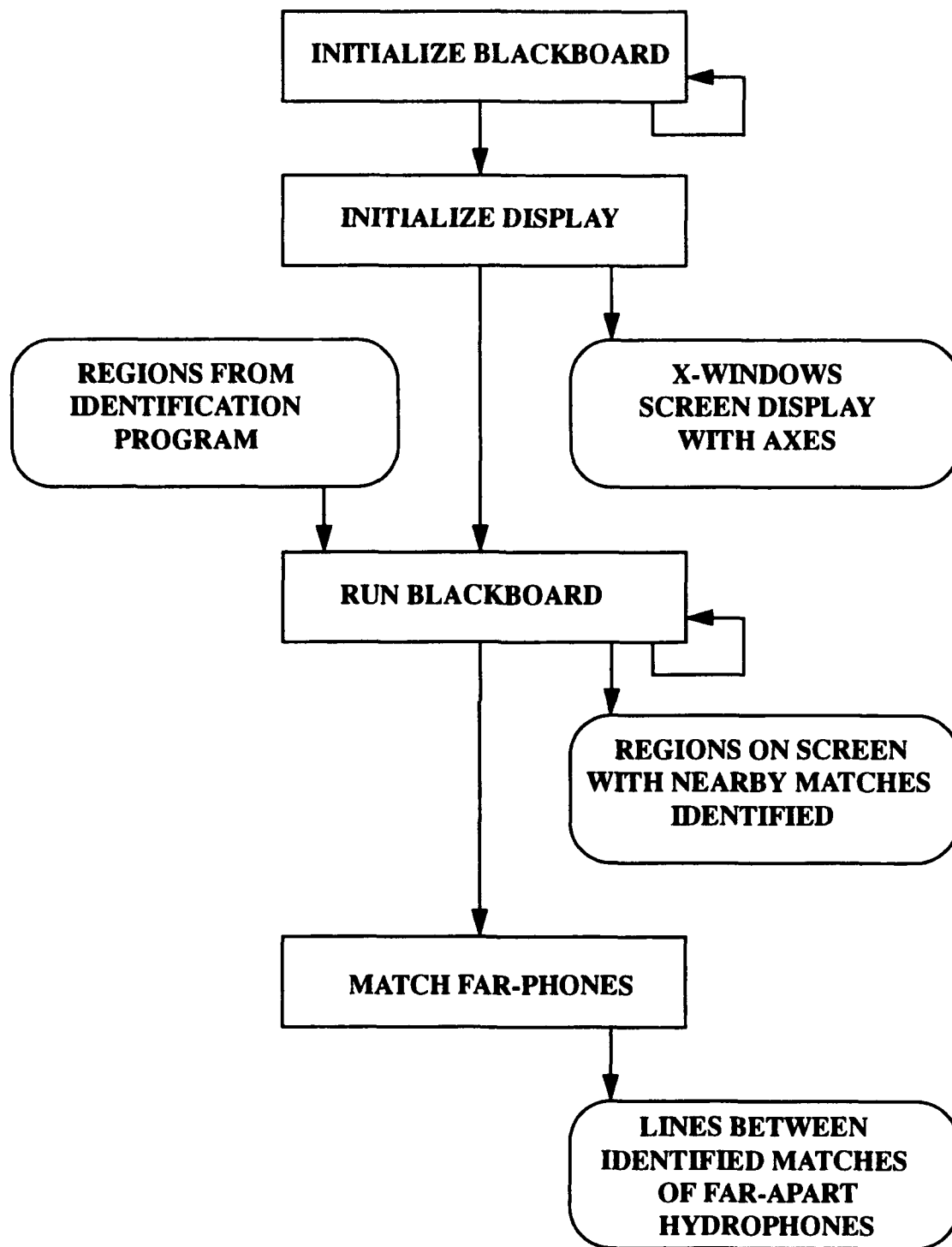
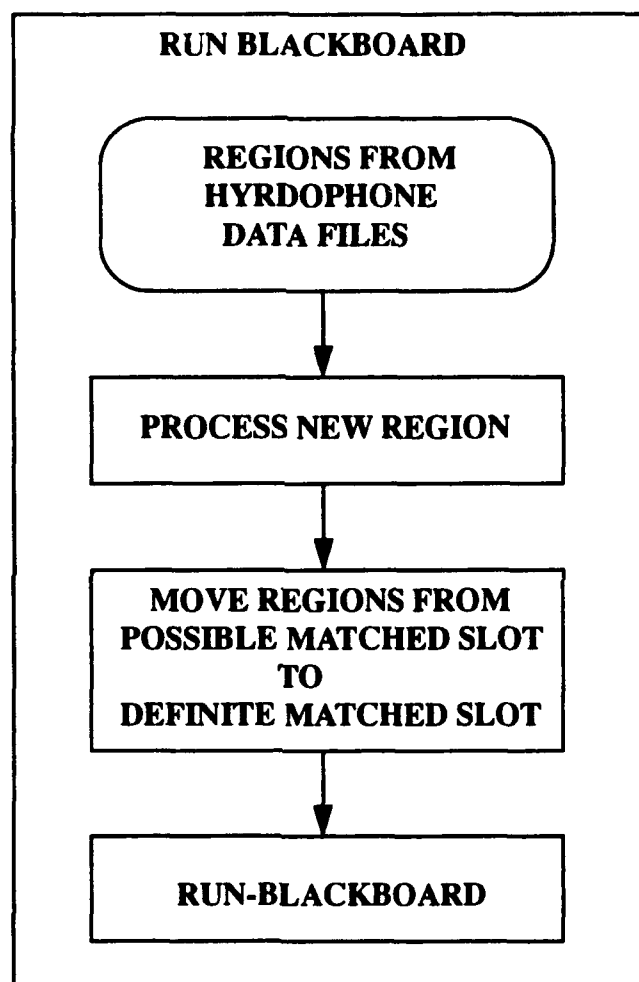
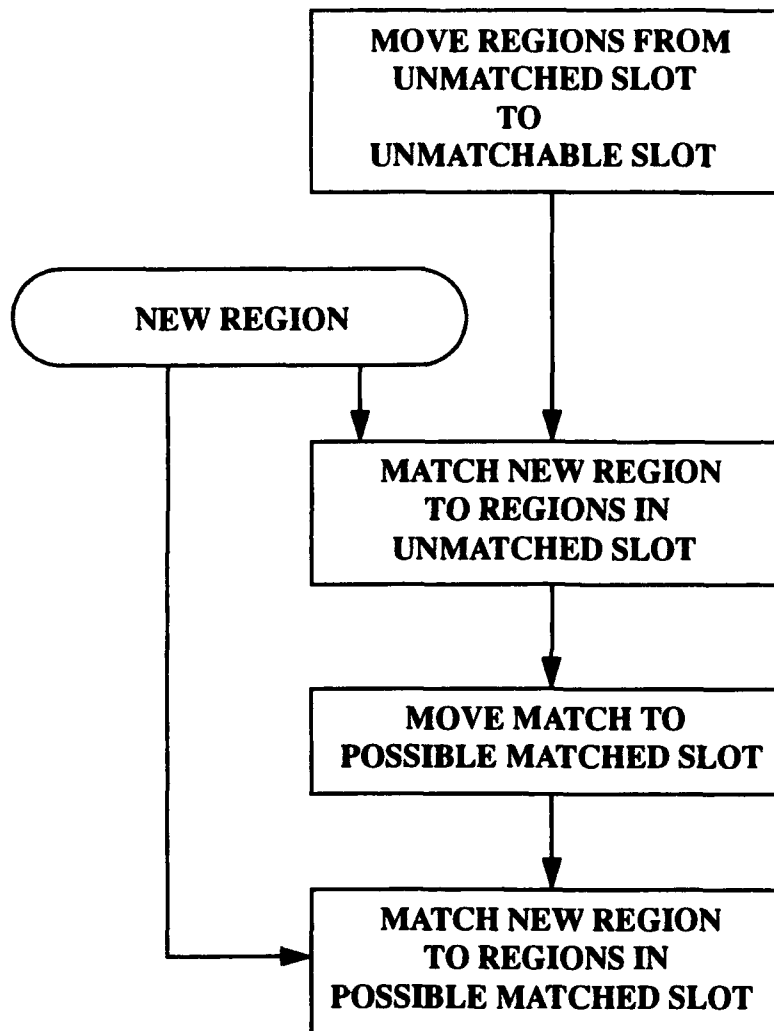


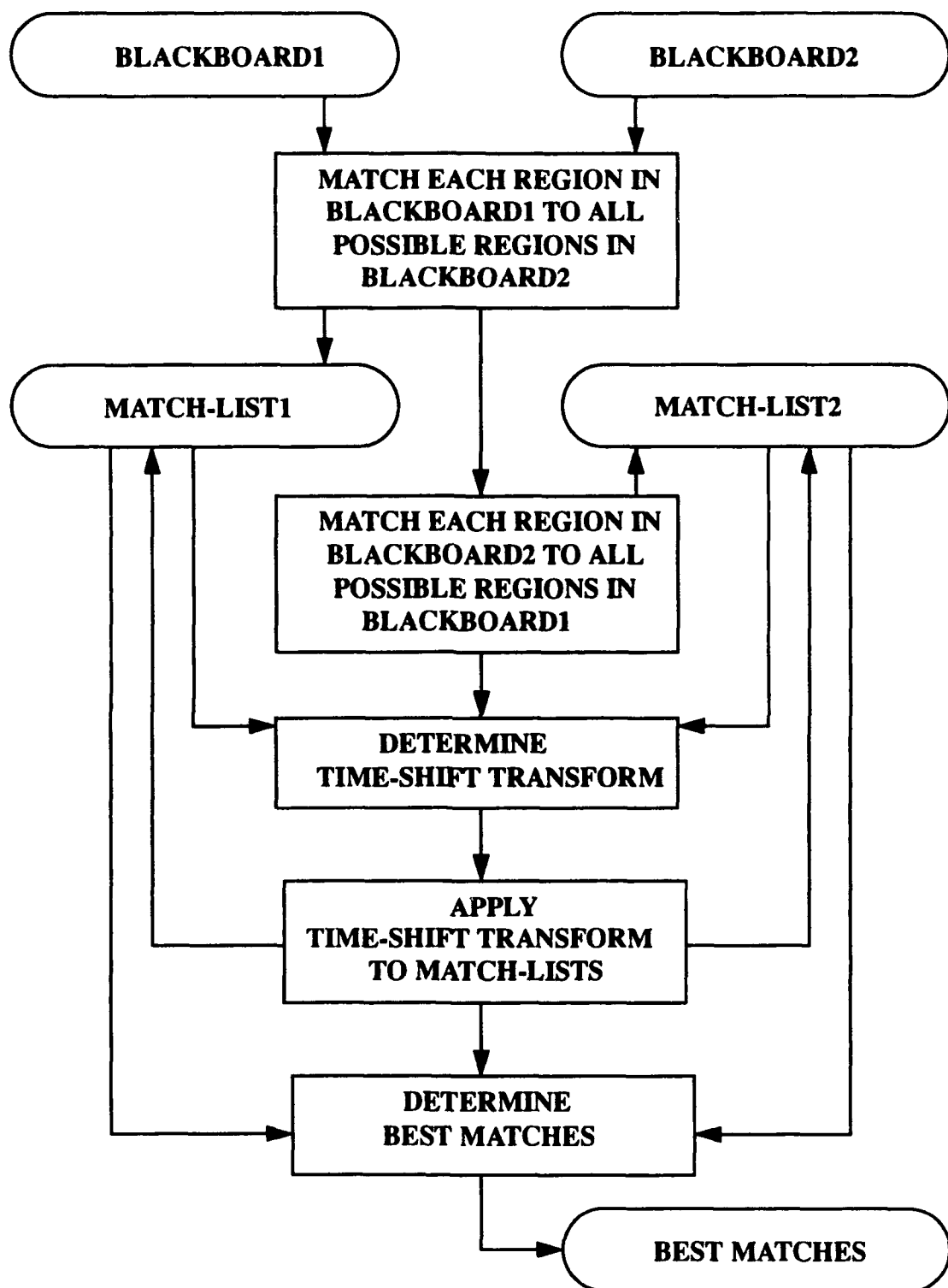
Figure 12: Diagram of Control Program.



**Figure 13: Diagram of Blackboard Controller.**



**Figure 14: Diagram of Process New Region Program.**



**Figure 15: Diagram of Match Far Regions Program.**



## **E. DATA STRUCTURES**

There are four classes of objects in the correlation program. They are: the blackboard class, the hydrophone class, the region class, and the screen class. Each time the program is run, there are two blackboard objects, two hydrophone objects for each blackboard object, many region objects, and one screen object.

The slots in the blackboard class are: phone1, phone2, screen-position, allowable-time-difference, unmatched-regions, definite-unmatched-regions, possible-matched-regions, definite-matched-regions, possible-region-matches, definite-region-matches, possible-region-match-time, and last-definite-matched-region-time. The blackboard class is a superclass of the hydrophone class through the phone1 and phone2 slots.

The screen position slot in the blackboard class is initialized with the blackboard and indicates whether the hydrophones are to be plotted on the upper part of the screen or the lower part of the screen. It remains constant for the life of the object. This slot is necessary because there are two blackboard objects in the program. The allowable time difference slot is also initialized with the blackboard and remains constant for the life of the object. It is determined from input parameters that state the relative positioning of the hydrophones, the assumed speed of sound in the deep sound channel, and an error factor due to uncertainty about the deep sound channel characteristics.

The unmatched regions slot in the blackboard class is a list of the region identifications of the regions that have not been matched to other regions and still fit within a time window where they might possibly be matched. The definite-unmatched-regions slot is a list of the region identifications of the regions that have not been matched to other regions and no longer fit within a time-window where they might possibly be matched.

The possible-matched-regions slot in the blackboard class is a list of the region identifications that have tentatively been matched to another region, but still fit within a time window where another region might possibly be matched to them. The definitely-matched-regions slot is a list of the region identifications that have been uniquely matched

to another region and no longer fit within a time window where another region might possibly be matched to them.

The possible-region-matches and definite-region-matches slots in the blackboard class are lists of lists of the region-identifications that are matched, combined with the goodness-of-fit value and the time delay between the region match. The possible-region-match-time and last-definite-region-match-time slots are used by the blackboard controller to determine whether it should implement the definite-matched-region knowledge source.

The hydrophone class is a sub-class of the blackboard class. It contains the following slots: file, blackboard-id, phone-id, latitude, and phone-time. The file slot is the name of the file that contains the region data for one hydrophone. It is supplied at the beginning of the program and remains constant for the life of the object. The blackboard-id slot corresponds to the screen-position slot from the blackboard class. The latitude slot is supplied as input to the initialization of the hydrophone and represents the relative position of the hydrophone to the other hydrophones. The phone-time slot corresponds to the start time of the most recently retrieved region from the input file.

The region class contains the following slots: blackboard-id, phone-id, region-id, region-type, pixels, time-min, time-max, freq-min, freq-max, and area. The blackboard-id slot corresponds to the screen-position slot in the blackboard class. The phone-id slot is set to the phone-id of the relevant phone object. The region-id slot is a unique machine-generated identification by which the region can be identified. The region-type slot is a letter corresponding to the identified type of region. The region type is the first element of the input list. The pixels slot is a list of time and frequency values for each pixel in the region. The pixels make up the rest of the input for a region after the region type. The time-min, time-max, freq-min, and freq-max slots are the extremes of time and frequency for the region. They are calculated after being read. The area slot is the number of pixels in the region.

The screen class has the following slots: id, borders, left, bottom, width, height, title, activate-p, lower-y-text-position, upper-y-text position. The id slot is the address of the

plot. The computer sends plotting commands to this address. The borders, left, bottom, width, and height slots in the screen class are for physical positioning of the plot on the graphics screen. The two text-position slots are for positioning text on the screen.

## **F. PROGRAM COMPONENTS**

The control program performs the initialization of the data structures and calls on the major controlling components. Two blackboard objects and a screen object are created and initialized. Each blackboard object initialization call includes the names of the two files that contain the region data from the nearby hydrophones, the relative position of these hydrophones in nautical miles, and the relative screen position to which the output will be plotted. The relative physical positions were called "latitude" in order to avoid confusion with the screen position of the hydrophone output. After the blackboard objects are initialized, a screen object is initialized. The initialization of the screen object causes the graphics screen to display a plot. Once this is done, the blackboard controller for the first blackboard is invoked. This causes the first two nearby hydrophones files to be read and their regions to be correlated and displayed. After the data set for the first set of hydrophones has been exhausted, the blackboard controller for the second blackboard is invoked. With two blackboard objects of nearby correlated hydrophones completed, the control program calls on the far-apart-hydrophone matching program to match the matched regions from the two blackboards.

When a blackboard object is created, the computer associates a name with the object and sets aside memory for that object to use. In the initialization of the blackboard, each of the two hydrophones are started. This involves opening their files for reading and adding the information about the hydrophone's latitude and screen position to the hydrophone data structure. The blackboard's screen position is added to its data structure, and the allowable time difference is determined from the latitude difference of each hydrophone divided by the assumed sound speed and multiplied by an uncertainty factor.

After the blackboard objects are initialized, the screen object is initialized. During the initialization of the screen object, the plot window is drawn to the screen, the lines delineating the minimum and maximum frequencies for the hydrophones are drawn, as are tic marks for each minute in time.

Once the initialization is done, the blackboard controller is called for each blackboard. The blackboard controller is a recursive program. It first checks the phone-time slot for each hydrophone and gets the next region from the hydrophone with the oldest time. The hydrophones stay synchronized in time by getting regions in this manner. After a region is retrieved from the hydrophone file, the controller calls on a program to process the new region. Once the process-new-region program returns, the controller checks the time difference between the last possible region match and the last definite region match against the allowable time difference. If sufficient time has passed, the controller calls the program for moving region matches from the possible-matched-regions slot to the definite-matched-regions slot. After this is done, the blackboard controller calls itself. This continues until both hydrophone files have been exhausted.

The process-new-region program starts by calling on a program to move regions from the unmatched-region slot to the unmatchable-regions slot. Regions are moved to the unmatchable-regions slot when they can no longer be matched to any new regions because the start times are too far apart. The process-new-region program then attempts to match the new region to all regions from the other hydrophone in the unmatched-region slot. A time-shift and overlap technique is employed for matching regions. The time element of each pixel in one region is modified so that it is time-correlated to the region to be matched, then a goodness-of-fit is determined from the amount of overlap between the two regions. If a match is made, both regions are moved to the possible-matched-regions slot. After the unmatched regions have been checked, an attempt is made to match the new region to all regions in the possible-matched regions slot.

After the control program calls the blackboard controller for each of the two pairs of nearby hydrophones, it calls the program for matching far apart hydrophones. The far-

match program first takes the list of definite-region-matches from one blackboard and matches each matched pair to all possible regions from the list of definite-region-matches from the other blackboard. This produces a list, for each nearby hydrophone pair, of far-apart region pairs with a goodness-of-fit and time-delay. The number of possible regions for matching is constrained by the travel time of sound between the two regions. After the far-apart matching is done for the regions from one blackboard, it is done for the other. Now there are two far-apart match lists, with each matched region from each pair of nearby hydrophones matched to all possible regions from the other set of nearby hydrophones with the goodness-of-fit and associated time delay. Each list of possible matches for each hydrophone pair is ordered, from best to worst, by the goodness-of-fit parameter. The time-shift transform is then determined from the matched lists. The time-shift transform is then applied to both matched lists. This modifies the goodness-of-fit numbers for each of the region matches for each far-apart pair of matches. This modifies the ordering of the list for each hydrophone, so each hydrophone is re-ordered, from best to worst, by the goodness-of-fit parameter. Once this is done, a get-best-matches program is called to determine the best matches between the two sets of far-apart hydrophones. It does this by finding all of the one-to-one correspondences between the best matches for each pair of regions. Lines are then drawn between the far-apart regions that have best matches. Once this is done, all of these regions are removed from the far-apart match lists. Then the get-best-matches is called again, followed by drawing lines between the additional best matches.

## **G. ABANDONED ALGORITHMS**

The region matching algorithm for the nearby regions originally attempted to limit the matchable time delay based on the assumed speed-of-sound through the water and the distance between the hydrophones. This did not work because of the distortions that regions could go through during the identification phase. Some regions would have their front ends removed because of the pixel trimming that goes on in the low-level vision processing. The

fix for this simply involved putting in artificial positions for the nearby hydrophones that were farther apart than was actually the case.

When attempting to match regions between the far apart hydrophones, the smallest region from the matched pairs of nearby regions was used for the time-shift and overlap comparison. This gave less satisfactory results than using the largest region from the matched pairs.

Knowledge sources in a blackboard architecture are usually independent and can frequently act in parallel. We attempted to implement the blackboard controller to take advantage of this independence by having the knowledge sources called up as separate processes. This would probably have worked if a multi-processing computer was available or if multiple computers were used. Unfortunately, the process controller grabbed most of the processing time, giving very little to the multiple processes that were running. This was mostly due to the fact that every time a pixel was drawn to the screen, an interrupt occurred, causing a context switch.

## **VII. DISCUSSION OF RESULTS**

### **A. INTRODUCTION**

The identification program was very successful in identifying t-phase signals. Most calibration signals and whale moans were also identified. There were a few false identification problems with whale moans and also some whale moans that were classified as unidentified signals.

The correlation program was successful in correlating regions to nearby hydrophones. It also successfully correlated regions to far apart hydrophones when all possible regions had been correlated to nearby hydrophones. False correlations tended to occur when the proper region on one hydrophone set was not available for correlation.

### **B. IDENTIFICATION PROGRAM**

The identification program was run on a Sun Sparcstation 10. Although over 10 hours of data for 16 hydrophones were provided for this project, only 2.2 hours of data for four hydrophones were used due to disk space limitations. The data set for each run included approximately 1120 seconds of Fourier transformed data from one hydrophone. This size data set was chosen so that full use could be made of the screen width on the workstation when displaying the results of the correlation program. The average running time was 33.8 minutes with a sample standard deviation of 13.7 minutes. The deviation is large because extremely noisy time periods caused the execution time to double over time periods that just contained whale moans. Execution time for a set which consisted exclusively of whale moans was approximately 21.7 minutes. The average memory use including program and data storage was 14 megabytes with a sample standard deviation of 2.4 megabytes. Memory usage for a set which consisted exclusively of whale moans was approximately 12.2 megabytes.

A comparison was made of the whale moans that were visually identifiable by a human and the whale moans that were identified by the program. There was one data set of 1120

seconds of sound that was strictly whale moans. During that data set, the program identified 91.7% of 156 human-identified whale moans that were received on four hydrophones. During periods of seismic activity and high noise, the number of identifiable moans was reduced for both the human observer and for the program. Over four data sets that showed both seismic activity and high noise covering a total of 4480 seconds, there were 469 human-identified whale moans, 68.9% of which were identified by the program.

The number of false identifications was negligible. There were two instances where a region was identified as a whale moan when it was actually two moans that slightly overlapped. There was also a problem of whale moans being mistakenly identified in the section of the hydrophone recording that was extremely noisy. Over a 15 minute period of noisy data, where nothing is identifiable by human vision on the frequency plots of two hydrophones, there were 30 whale moans identified. The few calibration signals that escaped elimination in the low-level processing phase of the identification program were also sometimes identified as whale moans.

There were many fewer t-phase signals in the data than whale moans, although t-phase signals tended to have a larger area. A single t-phase signal often consisted of many regions. Also, a t-phase signal can last for several minutes. One signal that was analyzed by the program lasted for 15 minutes across two data sets. All t-phase signals were properly identified by the program. There were a limited number of unidentified regions in the data that might have been associated with a t-phase signal, but in every case, there were several small regions near these unidentified regions that were identified as t-phase signals. Also, the author could not be certain that these regions were actually a part of the t-phase signal.

Most of the calibration signals were removed in the low-level vision processing phase of the program. There was a small number of calibration signals that made it to the high-level processing and these were identified as unknown signals or whale moans.

Signals that were not classified as whale moans, calibration signals, or t-phases, were classified as unknown. The identifiable whale moans that were not classified as such were all classified as unknown. Also, unknown signals were often small parts of whale moans



that didn't get picked up in the same group as the rest of the signal. During the high-noise section of the hydrophone recording, there were many small regions of noise that were classified as unknown.

### C. CORRELATION PROGRAM

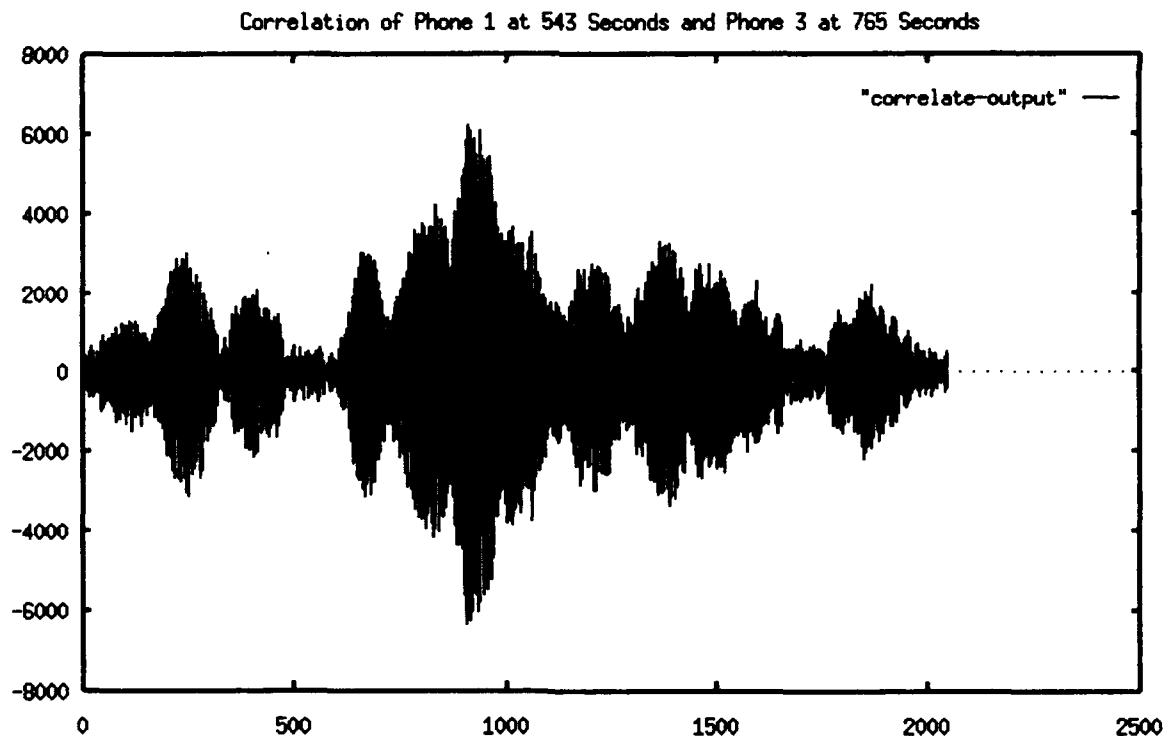
The correlation program was run on a Sun Sparcstation 10. Regions covering approximately 1120 seconds of hydrophone data for each of four hydrophones were input on each run. The average CPU time for the correlation program was approximately 28 minutes.

Correlations of signals between near-field hydrophones are easily identifiable by human inspection. In the first data segment, which consists entirely of whale moans (see Figure 11), there should be 86 near-field matches. The program determined 81 (94.1%) of those matches. Similar results were obtained for all other data segments where there were whale moans to be correlated.

The results were less satisfactory for t-phase signals. The identification program tended to break t-phase signals up into several different regions. Often, the manner in which these regions were determined was different for the near-field hydrophones. Over the span of seven data sets, covering 2 hours and 10 minutes, there were 365 regions identified as t-phase signals. Near-field correlations of these t-phase signals dropped to 42.7%. Part of the reason for this was that most of the programming effort concentrated on the correlation of whale moans.

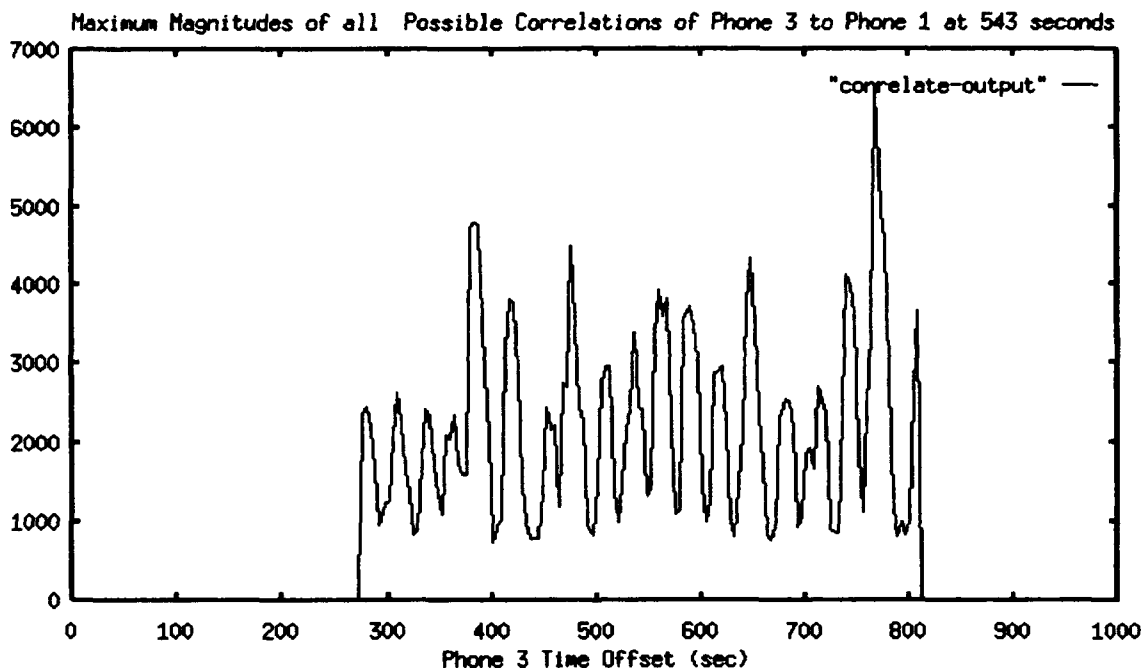
The symbolic correlation of signals to the far-field hydrophones is not obvious to the human observer. When looking at a plot of the signals without the far-field correlations connected, one can only visually correlate one or two whale moans per data set. After the far-field correlations are drawn, it is easier to see that the correlations drawn by the program are probably correct. This human verification after-the-fact is not good enough to claim that the far-field correlations are correct. Another approach was used to verify the program output.

For each pair of near-field correlated whale moans in the first data set, a Fast Fourier Transform correlation covering a 16 second time period was run against time delays for the possible matches to the far-field hydrophones. Figure 16 shows a correlation of a whale moan between two hydrophones with a sample data set covering 16 seconds. The correlation is between far-apart hydrophones in Figure 11(a) at the indicated times. Phone 1 is the lower hydrophone in the lower set of hydrophones and Phone 3 is the lower hydrophone in the upper set of hydrophones. The vertical axis is the magnitude of the correlation. The horizontal axis is in units of 1/128 seconds. The correlation is for eight seconds on either side of the selected time offsets. The correlation at the exact time offset is at the 1024 value point on the horizontal axis. The maximum correlation occurs slightly less than a second prior to that point and has a magnitude of approximately 6000. The two time offsets in this graph were selected because they match time offsets for a correlation chosen by the shape correlation program.



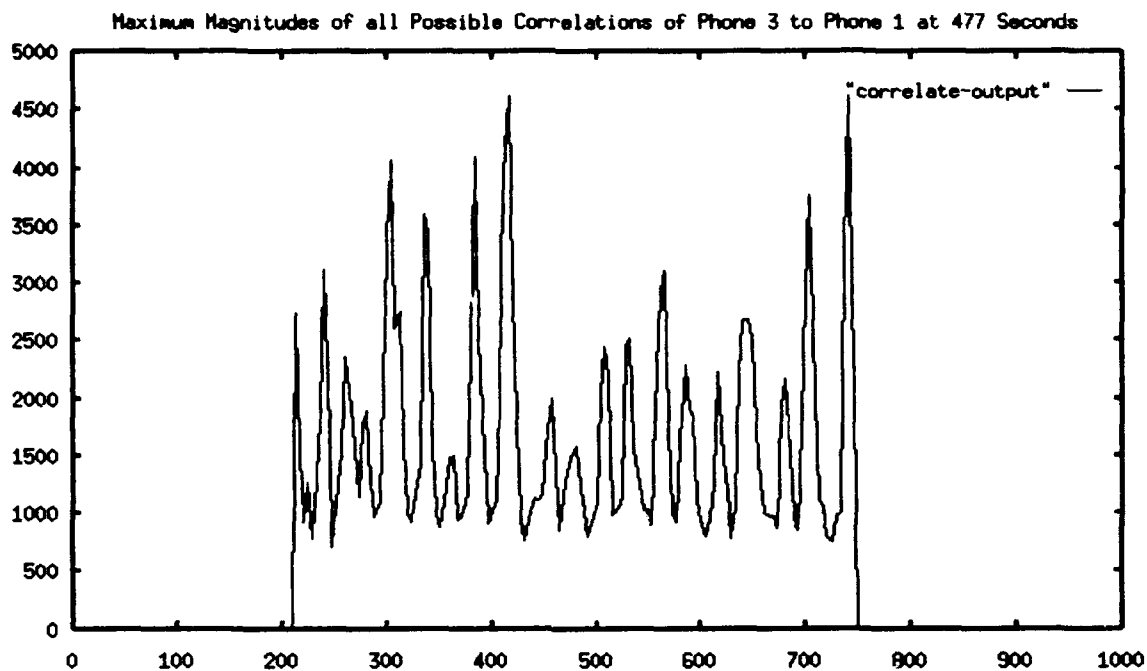
**Figure 16: Far-Field correlation of a Whale Moan.**

Sixteen-second FFT correlations were calculated every four seconds over the entire possible range of correlations for each whale moan in the first data set. The maximum magnitude within that correlation was then determined and saved. The maximum values for the correlations over the range of possible correlations were plotted. Figure 17 shows a plot of the maximum correlations of Phone 3 to Phone 1 at the 543 second time offset for Phone 1. The maximum of these maximum-correlation values occurs at approximately the 765 second time offset for Phone 3, as shown in Figure 16. This verifies the correlation chosen by the symbolic correlation program for this particular whale moan.



**Figure 17: Maximum Far-Field Correlation Magnitudes.**

In the majority of the cases, these correlations provided verification that the symbolically chosen set of far-field correlations were correct. When the maximum correlation versus time plot was presented, there were sometimes peaks of similar size that would indicate that the correlation could be to any of two or three different whale moans. Figure 18 is a plot of maximum far-field correlations that have an ambiguous best correlation.



**Figure 18: Maximum Far-Field Correlation Magnitudes.**

The FFT correlations validated the signal shape correlation algorithm. In the data set shown in Figure 11(a), there are 29 far-field whale-moan correlations. FFT correlations were calculated for each of the moans and the results were compared to the results of the shape correlation program. Of the 29 shape correlations, there were 19 moans where at least one FFT correlation between Phones 1 and 2 and Phones 3 and 4 had a maximum correlation at the same time offset. Twenty-five of the 29 moan correlations had a peak, corresponding to the time offset chosen by the shape correlation program, in the FFT far-field correlation magnitudes which was among the top three peaks.

There were occasional errors made by the correlation program. The program did not attempt to correlate regions between far-field hydrophones if a near-field correlation did not exist. This sometimes caused a false correlation between far-field hydrophones. False correlations also took place at the end of the data set. The whale moans from the lower set of hydrophones consistently correlated to moans from the upper set of hydrophones with a large time offset. At the end of the data set, the moans that should have been correlated were

correlated for the upper hydrophones and there were moans in the lower set of hydrophones that would obviously correlate to moans from the upper set of hydrophones in the next data set. Without the next data set available, some of the uncorrelated moans from the lower set of hydrophones correlated moans from the upper set of hydrophones that should have remained uncorrelated.

Almost all of the valid correlations of whale moans between the far-apart hydrophones showed a consistent time offset of approximately 240 seconds. This is near the limit of the possible straight-line travel time of sound between these two hydrophone pairs. The lower set of hydrophones shown in Figure 11 is south of the upper set of hydrophones. This indicates that the whale moans were coming from the south.

When performing the far-field correlations of whale moans, the whale moans often had a goodness-of-fit value of over 0.7. This indicated that there was very little attenuation of the signal over the approximate 335 km distance between the hydrophone pairs. It also suggests that these low frequency whale moans may be coming from distances of many hundreds of kilometers south of the southern pair of hydrophones.

## VIII. CONCLUSION

### A. INTRODUCTION

The goals of this thesis have been met. The computer vision model properly identifies t-phase signals, whale moans, and calibration signals. The correlation program properly correlates whale moans between near-field and far-field hydrophones.

These programs are not production quality programs. They are written in prototype languages and improvements could be made in the data structures and in the method of execution. However, they can serve as prototype programs for incorporation of the ideas presented into other programs.

### B. ACHIEVEMENTS

There were major accomplishments of this thesis that can be of significant benefit to the research community. The first is that a method has been developed for unambiguously identifying t-phase signals received at submarine hydrophones. This has been a problem for *geophysicists studying these signals*. The other major accomplishment of this thesis is that a method has been developed for correlating signal shapes between both near-field and far-field hydrophones. This method is particularly applicable to the whale research community.

Geophysicists studying seismic events in the northeast Pacific ocean basin have a set of software tools that allow them to scroll hydrophone data across the screen of a graphics workstation so that they can look for t-phase signals. By taking the ideas from this thesis, they can add some modules to that software and develop a reliable t-phase detector. There is also the possibility of using the identification techniques to develop an acoustic tsunami warning system. The ideas from the correlation program could be used to improve the current correlation techniques.

This thesis demonstrates a "proof-of-concept" for whale researchers that are interested in remotely tracking the migration paths of large open-ocean baleen whales. These whales are extremely difficult to monitor using traditional survey methods. A

method is presented here for automatically identifying whale moans and correlating them to both near-field and far-field hydrophones.

### **C. WEAKNESSES**

The algorithm for the identification program turns on pixels in the low level processing based on a moving threshold. There is a target percentage of pixels that the algorithm wants turned on for each second in order to be able to develop distinct regions. This is sometimes a detriment when there is both an intense t-phase signal and whale moans occurring simultaneously. Often the whale signals lose a lot of their shape because their pixels move to the t-phase signal.

In order for a program to effectively work with this data in a production fashion, it must process it in a real-time mode. The identification program does not work in this manner. An arbitrarily large section of data is handled at one time by the program. That entire section of data is manipulated several times during the low-level vision-processing phase of the program. The high-level vision-processing phase also performs each step with a view to the entire data set. The real-time quality of the software was not a concern during this thesis since the programs were not intended to be production programs.

Another weakness of the programs written for this thesis is their speed. It takes approximately 2.5 hours of computation time to produce a single plot containing 20 minutes of data. Since the programs were not intended as production programs, speed was not a concern. The incorporation of the ideas in this thesis into a production program can be done without a significant performance penalty.

Finally, the methods used to do the symbolic correlation should be verified by use of Fourier transform correlation techniques. These techniques would also give a much more accurate time delay so that lines-of-position can be precisely determined.

## APPENDIX A: SAMPLE OF RESULTS FROM IDENTIFICATION PROGRAM

```
group(1,[3,2,1]).
group(2,[4]).
group(3,[5]).
group(4,[6]).
group(5,[7]).
group(7,[9]).
group(6,[12,10,11,8]).
group(8,[13]).
group(9,[15,14]).
group(10,[17,16]).
group(11,[18]).
group(12,[21,20,19]).
group(13,[22]).
group(14,[23]).
group(15,[24]).
group(16,[25]).
group(17,[26]).
group(19,[28]).
group(18,[32,29,30,27]).
group(20,[31]).
group(21,[34,33]).
group(22,[38,36,37,35]).
group(23,[40,39]).
group(24,[41]).
group(25,[42]).
group(26,[44]).
group(27,[45]).
group(28,[47]).
group(29,[46,48,43]).
group(30,[49]).
group(31,[53,50]).
group(32,[51]).
group(33,[52]).
group(34,[54]).
group(35,[55]).
group(36,[56]).
group(37,[57]).
group(38,[58]).
group(39,[59]).
group(40,[61,60]).
group(41,[62]).
group(42,[63]).
group(43,[64]).
group(44,[65]).
group(45,[66]).
group(46,[67]).
group(47,[68]).
group(48,[69]).
group(49,[70]).
```



```

group(50,[71]).
group(51,[72]).
group(52,[73]).
group(53,[76,74]).
group(54,[75]).
group(55,[83,81,77]).
group(56,[78]).
group(57,[82,80,79]).
group(58,[84]).
group(59,[85]).
group(60,[88,86]).
group(61,[87]).
group(62,[91,90,89]).
group(63,[92]).
group(64,[94,93]).
group(65,[95]).
group(66,[98,97,96]).
group(67,[99]).
group(68,[102,101,100]).
group(69,[103]).
group(70,[104]).
group(71,[105]).
group(72,[108,106]).
group(73,[107]).
group(74,[109]).
group(75,[110]).
group(76,[112,111]).
group(77,[114]).
group(78,[115,116,113]).
group(79,[118,117]).
group(80,[119]).
group(81,[121,120]).
group(82,[122]).
group(83,[123]).
group(84,[125,124]).
group(85,[126]).
group(86,[127]).
group(87,[128]).
group(88,[129]).
group(89,[130]).
group(90,[131]).
group(91,[132]).
group(92,[133]).
group(93,[134]).
group(94,[135]).
group(95,[136]).
group(96,[137]).
group(97,[138]).
group(98,[139]).
group(99,[140]).
group(100,[141]).
group(101,[142]).
group(102,[144,143]).

group_qualities(1,32,61,1,19,147,66,25).

```

group\_qualities(2,12,14,22,22,3,13,100).  
group\_qualities(3,43,60,25,34,87,54,48).  
group\_qualities(4,12,15,29,30,4,21,50).  
group\_qualities(5,33,35,38,38,3,17,100).  
group\_qualities(6,37,62,39,60,183,55,31).  
group\_qualities(7,9,13,40,42,8,27,53).  
group\_qualities(8,24,28,59,63,13,36,52).  
group\_qualities(9,36,62,59,73,121,53,29).  
group\_qualities(10,28,51,74,88,121,58,33).  
group\_qualities(11,55,62,85,89,22,57,55).  
group\_qualities(12,37,62,93,107,139,48,35).  
group\_qualities(13,34,61,109,131,175,59,27).  
group\_qualities(14,35,61,132,146,148,57,36).  
group\_qualities(15,14,17,144,145,6,35,75).  
group\_qualities(16,34,60,148,156,89,49,36).  
group\_qualities(17,8,12,151,151,5,15,100).  
group\_qualities(18,24,59,159,173,133,67,24).  
group\_qualities(19,54,55,159,159,2,62,100).  
group\_qualities(20,8,10,173,173,3,12,100).  
group\_qualities(21,40,62,175,187,111,66,37).  
group\_qualities(22,29,62,186,201,130,65,23).  
group\_qualities(23,43,62,206,217,79,35,32).  
group\_qualities(24,37,58,223,236,118,73,38).  
group\_qualities(25,12,14,229,229,3,28,100).  
group\_qualities(26,61,62,236,236,2,29,100).  
group\_qualities(27,7,11,237,239,8,50,53).  
group\_qualities(28,8,10,254,255,5,29,83).  
group\_qualities(29,22,62,236,266,298,64,23).  
group\_qualities(30,13,16,260,260,4,27,100).  
group\_qualities(31,34,62,266,286,179,60,29).  
group\_qualities(32,7,10,280,280,4,26,100).  
group\_qualities(33,11,14,286,286,4,12,100).  
group\_qualities(34,39,43,290,293,8,46,40).  
group\_qualities(35,47,62,293,304,81,46,42).  
group\_qualities(36,27,29,308,309,5,29,83).  
group\_qualities(37,40,58,308,314,70,76,52).  
group\_qualities(38,7,9,310,310,3,13,100).  
group\_qualities(39,47,53,318,318,7,100,100).  
group\_qualities(40,57,62,318,322,13,48,43).  
group\_qualities(41,50,62,325,334,63,59,48).  
group\_qualities(42,14,16,333,333,3,31,100).  
group\_qualities(43,35,38,339,341,8,35,66).  
group\_qualities(44,47,50,339,339,4,8,100).  
group\_qualities(45,58,60,341,342,6,28,100).  
group\_qualities(46,13,15,346,347,5,31,83).  
group\_qualities(47,35,60,348,360,132,56,39).  
group\_qualities(48,7,10,359,360,4,13,50).  
group\_qualities(49,40,62,364,373,91,58,39).  
group\_qualities(50,10,12,370,371,5,35,83).  
group\_qualities(51,32,59,376,389,124,55,31).  
group\_qualities(52,9,13,386,388,7,23,46).  
group\_qualities(53,38,62,393,411,170,73,35).  
group\_qualities(54,55,57,397,397,3,8,100).  
group\_qualities(55,41,60,414,445,209,82,32).  
group\_qualities(56,32,34,416,417,4,17,66).

group\_qualities(57,23,40,423,445,125,99,30).  
 group\_qualities(58,24,28,450,450,5,144,100).  
 group\_qualities(59,49,55,450,450,7,105,100).  
 group\_qualities(60,24,62,457,470,145,72,26).  
 group\_qualities(61,49,56,457,458,11,112,68).  
 group\_qualities(62,33,62,476,495,178,60,29).  
 group\_qualities(63,61,62,499,499,2,16,100).  
 group\_qualities(64,39,62,503,524,129,55,24).  
 group\_qualities(65,31,35,522,525,17,35,85).  
 group\_qualities(66,39,57,526,535,51,39,26).  
 group\_qualities(67,47,61,538,547,52,50,34).  
 group\_qualities(68,37,62,555,571,121,60,27).  
 group\_qualities(69,34,62,574,588,162,58,37).  
 group\_qualities(70,31,61,591,602,154,50,41).  
 group\_qualities(71,36,62,611,620,143,58,52).  
 group\_qualities(72,29,61,625,646,200,55,27).  
 group\_qualities(73,53,55,625,625,3,14,100).  
 group\_qualities(74,32,50,652,659,83,56,54).  
 group\_qualities(75,52,61,660,666,34,52,48).  
 group\_qualities(76,42,62,675,688,112,66,38).  
 group\_qualities(77,33,35,689,689,3,17,100).  
 group\_qualities(78,37,61,688,706,141,45,29).  
 group\_qualities(79,44,62,709,729,129,48,32).  
 group\_qualities(80,34,42,732,741,53,38,58).  
 group\_qualities(81,47,60,743,746,17,28,30).  
 group\_qualities(82,31,39,752,754,19,51,70).  
 group\_qualities(83,51,54,754,754,4,12,100).  
 group\_qualities(84,45,57,769,778,49,52,37).  
 group\_qualities(85,60,61,781,781,2,13,100).  
 group\_qualities(86,3,10,782,782,8,5,100).  
 group\_qualities(87,54,62,795,799,20,42,44).  
 group\_qualities(88,11,13,801,801,3,24,100).  
 group\_qualities(89,3,14,808,810,25,27,69).  
 group\_qualities(90,3,5,821,821,3,6,100).  
 group\_qualities(91,12,14,821,821,3,11,100).  
 group\_qualities(92,53,62,833,836,17,30,42).  
 group\_qualities(93,7,14,836,847,46,35,47).  
 group\_qualities(94,19,22,838,840,6,29,50).  
 group\_qualities(95,32,37,840,844,25,64,83).  
 group\_qualities(96,40,62,843,858,144,66,39).  
 group\_qualities(97,23,28,868,868,6,7,100).  
 group\_qualities(98,51,54,868,868,4,21,100).  
 group\_qualities(99,30,40,872,878,45,50,58).  
 group\_qualities(100,52,54,872,872,3,43,100).  
 group\_qualities(101,43,53,880,882,16,38,48).  
 group\_qualities(102,52,54,1118,1118,2,27,66).  
  
 unknown\_group(1,3,[5]).  
 unknown\_group(4,16,[25]).  
 unknown\_group(5,21,[34,33]).  
 unknown\_group(6,23,[40,39]).  
 unknown\_group(7,34,[54]).  
 unknown\_group(8,35,[55]).  
 unknown\_group(9,37,[57]).  
 unknown\_group(10,40,[61,60]).

```

unknown_group(11,41,[62]).
unknown_group(12,43,[64]).
unknown_group(13,47,[68]).
unknown_group(14,49,[70]).
unknown_group(15,61,[87]).
unknown_group(16,65,[95]).
unknown_group(17,66,[98,97,96]).
unknown_group(18,67,[99]).
unknown_group(19,70,[104]).
unknown_group(20,71,[105]).
unknown_group(21,74,[109]).
unknown_group(22,75,[110]).
unknown_group(23,80,[119]).
unknown_group(24,81,[121,120]).
unknown_group(25,82,[122]).
unknown_group(26,84,[125,124]).
unknown_group(27,87,[128]).
unknown_group(28,92,[133]).
unknown_group(30,99,[140]).
unknown_group(31,101,[142]).

best_line_fit(1,3.92699E-
01,8.310769330205034E+00,8.936860787545353E+01,1.96686578554254
6E+00,5.504832046121114E+00).
best_line_fit(2,0.0E+00,6.666666666666666E-
01,6.666666666666666E-01,0.0E+00,0.0E+00).
best_line_fit(3,3.92699E-
01,4.864060736807142E+00,2.995710082833223E+01,1.35660598015771
3E+00,2.604220836109694E+00).
best_line_fit(4,2.748893E+00,1.442320293329162E+00,1.8210589592
88522E+00,3.467679539997935E-01,1.789105231333532E-01).
best_line_fit(5,0.0E+00,6.666666666666666E-
01,6.666666666666666E-01,0.0E+00,0.0E+00).
best_line_fit(6,7.85398E-
01,5.99266765641066E+00,4.947743342772709E+01,3.968161666771911
E+00,2.227869489847912E+01).
best_line_fit(7,3.92699E-
01,1.24022124221869E+00,1.993488418904624E+00,5.556897818880131
E-01,3.971365810953754E-01).
best_line_fit(8,1.178097E+00,1.340075183857666E+00,2.3461016394
72814E+00,7.994335220519558E-01,8.717344091464707E-01).
best_line_fit(9,3.92699E-
01,5.873938595058509E+00,4.837144438620045E+01,1.39121800810705
E+00,2.687877872530698E+00).
best_line_fit(10,3.92699E-
01,6.20814414758197E+00,5.291545010879767E+01,1.732849949248802
E+00,4.44991523848645E+00).
best_line_fit(11,3.92699E-
01,2.044337037890678E+00,5.38513338689388E+00,7.307223795709038
E-01,7.255916318889889E-01).
best_line_fit(12,3.92699E-
01,5.742931453477842E+00,4.369848090935798E+01,1.80455897930269
E+00,4.565103761438595E+00).
best_line_fit(13,3.92699E-
01,8.073903502210806E+00,8.571803842627608E+01,2.00817743376957

```

```

4E+00,6.492546576639387E+00).
best_line_fit(14,3.92699E-
01,7.658651150063783E+00,7.248707109104529E+01,1.69304840463405
9E+00,4.098832939811693E+00).
best_line_fit(15,3.92699E-01,9.725690848498098E-
01,1.148590076431551E+00,4.286549390719898E-
01,2.125210346795603E-01).
best_line_fit(16,3.92699E-
01,4.57225552928752E+00,3.320741577122268E+01,1.421099392410706
E+00,2.859531853424639E+00).
best_line_fit(17,0.0E+00,1.2E+00,2.0E+00,0.0E+00,0.0E+00).
best_line_fit(18,3.92699E-
01,9.255047910040037E+00,1.113076501467854E+02,2.56958253805543
E+00,1.017420361986031E+01).
best_line_fit(19,0.0E+00,5.0E-01,2.5E-01,0.0E+00,0.0E+00).
best_line_fit(20,0.0E+00,6.666666666666666E-
01,6.666666666666666E-01,0.0E+00,0.0E+00).
best_line_fit(21,3.92699E-
01,5.476144549077222E+00,4.070315981938675E+01,1.48722308328584
7E+00,3.025641297288723E+00).
best_line_fit(22,3.92699E-
01,7.712097045472777E+00,8.733482516882495E+01,2.04981068403393
3E+00,6.826388759258818E+00).
best_line_fit(23,3.92699E-
01,4.651269405339635E+00,3.305360740266237E+01,1.52106413728622
4E+00,3.204781893113417E+00).
best_line_fit(24,3.92699E-
01,4.887628702434714E+00,3.31452076662906E+01,1.723438095964227
E+00,4.327691218210757E+00).
best_line_fit(25,0.0E+00,6.666666666666666E-
01,6.666666666666666E-01,0.0E+00,0.0E+00).
best_line_fit(26,0.0E+00,5.0E-01,2.5E-01,0.0E+00,0.0E+00).
best_line_fit(27,2.748893E+00,1.323083134483586E+00,2.146587578
914547E+00,4.585569334063936E-01,2.641265631099703E-01).
best_line_fit(28,0.0E+00,6.399999999999999E-
01,5.599999999999999E-01,4.800000000000011E-01,2.4E-01).
best_line_fit(29,7.85398E-
01,1.033834400904913E+01,1.498501275177333E+02,3.45453041995935
8E+00,1.840361163311271E+01).
best_line_fit(30,0.0E+00,1.0E+00,1.25E+00,0.0E+00,0.0E+00).
best_line_fit(31,7.85398E-
01,6.53511165006327E+00,6.246741985786708E+01,2.069677339906318
E+00,6.002532424575429E+00).

horizontal_line(13,35,109,120).
horizontal_line(18,30,159,170).
horizontal_line(57,24,427,445).

whale(1,1,[3,2,1]).
whale(2,6,[12,10,11,8]).
whale(3,9,[15,14]).
whale(4,10,[17,16]).
whale(5,12,[21,20,19]).
whale(6,13,[22]).
whale(7,14,[23]).

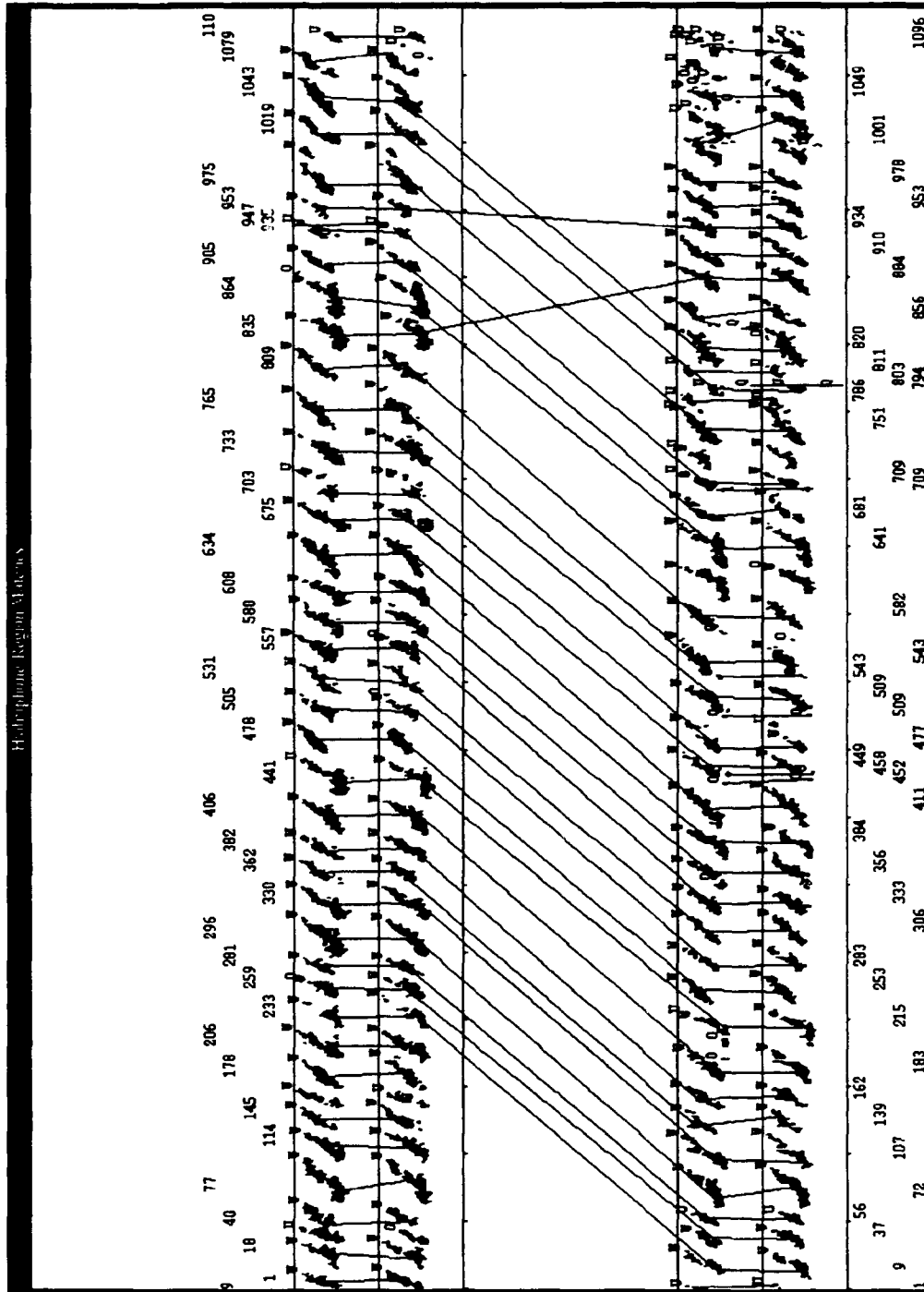
```

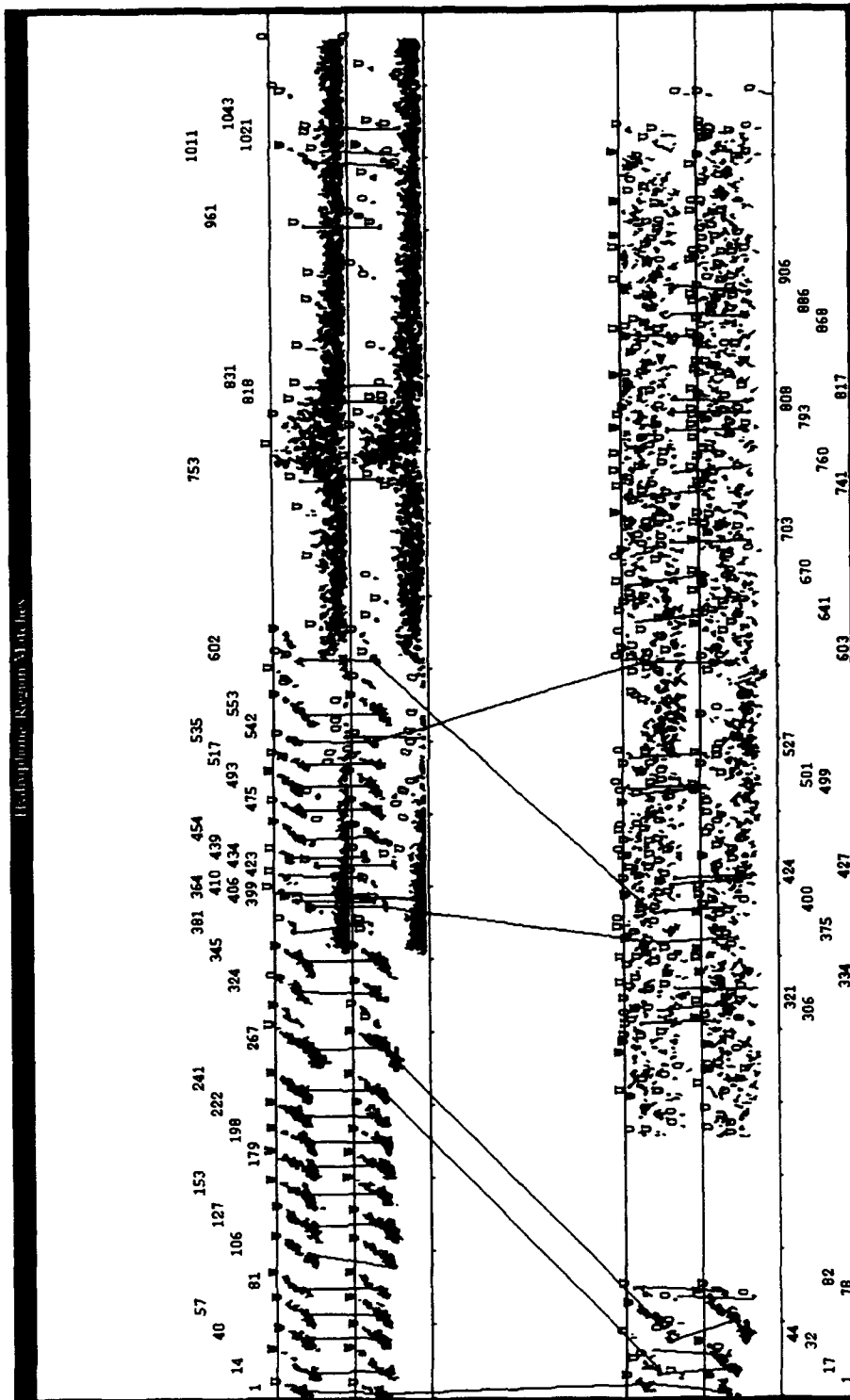
```
whale(8,18,[32,29,30,27]).
whale(9,22,[38,36,37,35]).
whale(10,24,[41]).
whale(11,29,[46,48,43]).
whale(12,31,[53,50]).
whale(13,51,[72]).
whale(14,53,[76,74]).
whale(15,55,[83,81,77]).
whale(16,57,[82,80,79]).
whale(17,60,[88,86]).
whale(18,62,[91,90,89]).
whale(19,64,[94,93]).
whale(20,68,[102,101,100]).
whale(21,69,[103]).
whale(22,72,[108,106]).
whale(23,76,[112,111]).
whale(24,78,[115,116,113]).
whale(25,79,[118,117]).
whale(26,96,[137]).
```

```
possible_quake(1,7,[9]).
possible_quake(2,27,[45]).
possible_quake(3,52,[73]).
possible_quake(4,86,[127]).
possible_quake(5,89,[130]).
possible_quake(6,93,[134]).
```

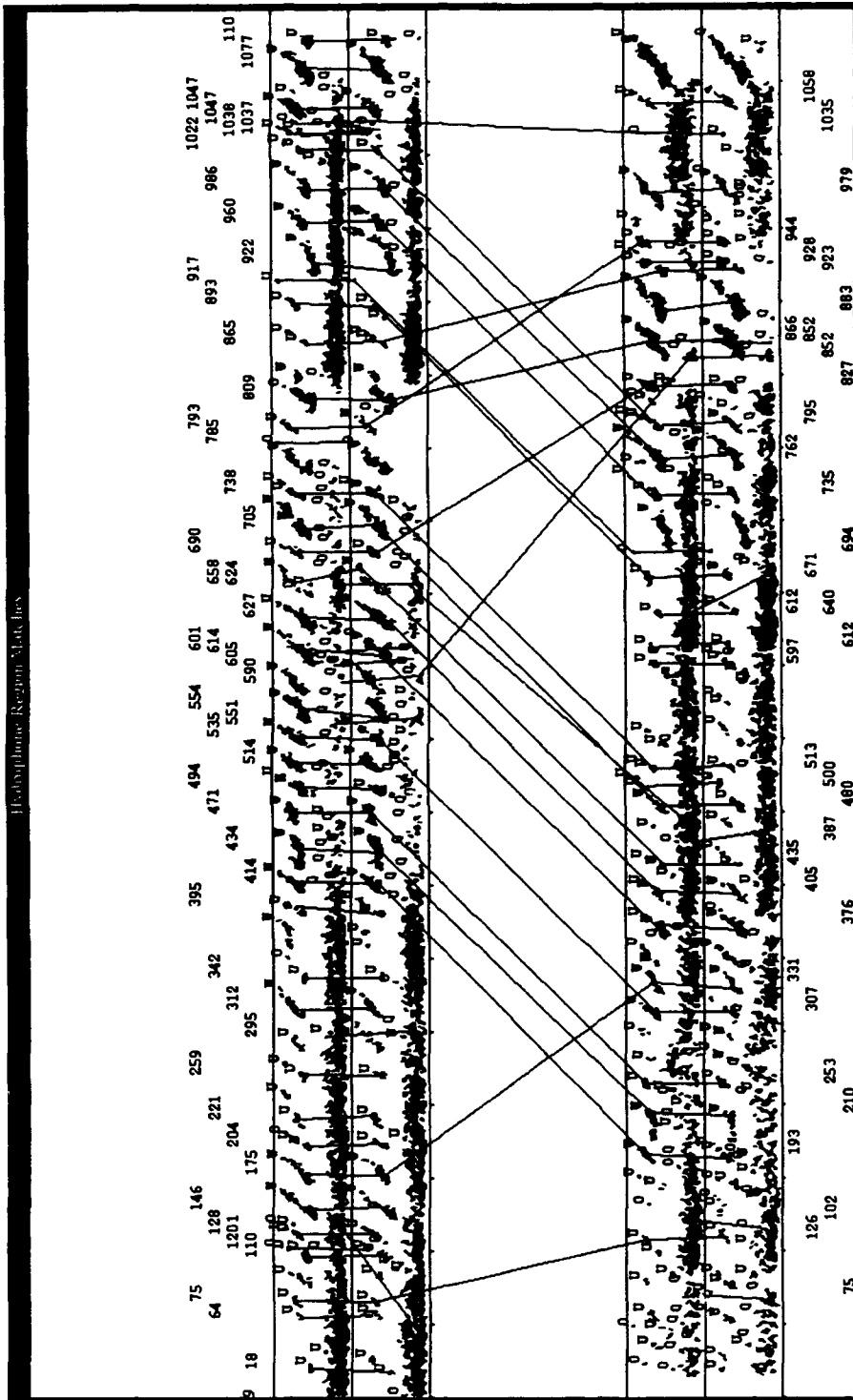
```
far_quake(1,[9]).
far_quake(2,[45]).
far_quake(3,[73]).
far_quake(4,[127]).
far_quake(5,[130]).
far_quake(6,[134]).
```

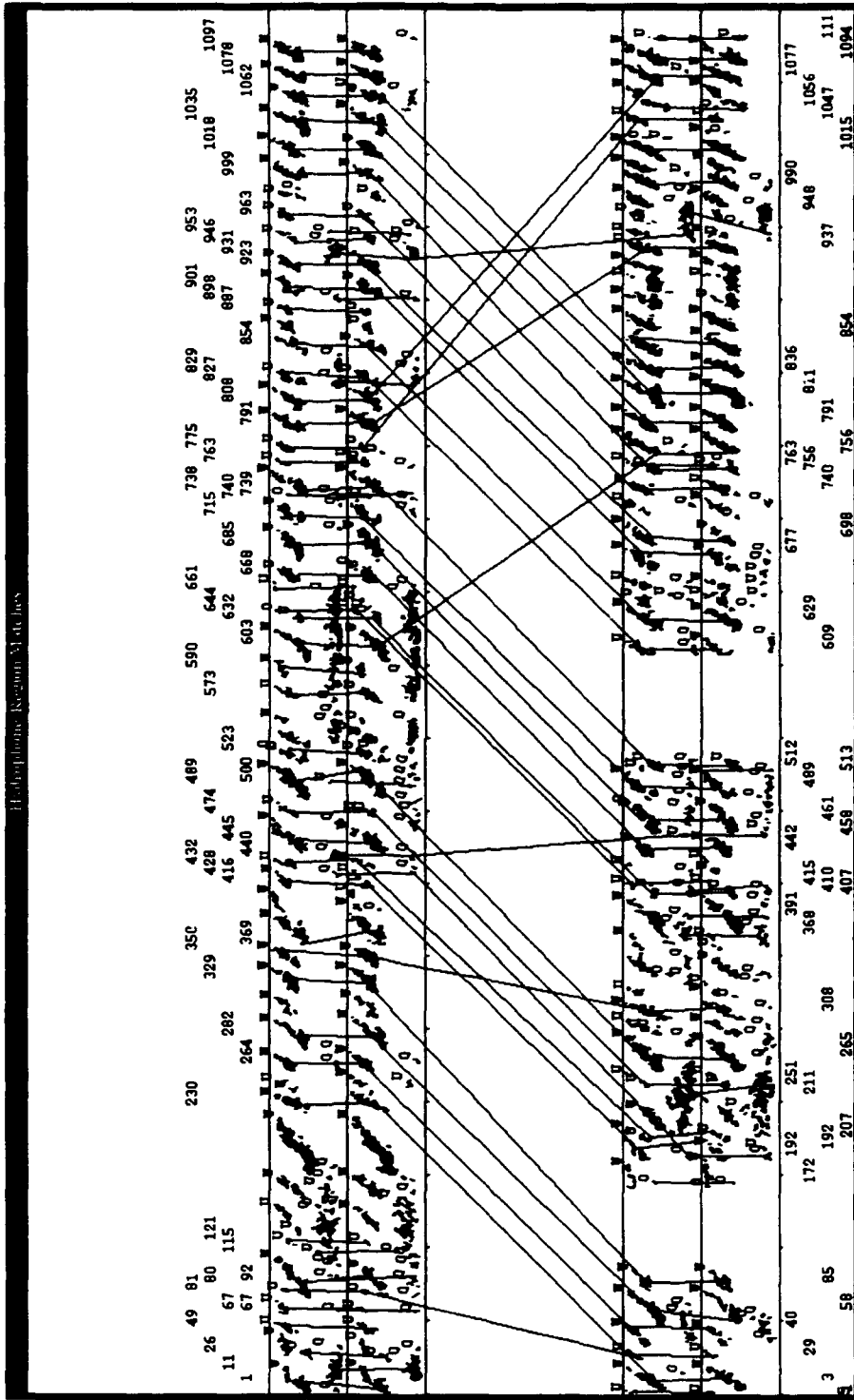
# APPENDIX B: PLOTS PRODUCED BY THE CORRELATION PROGRAM

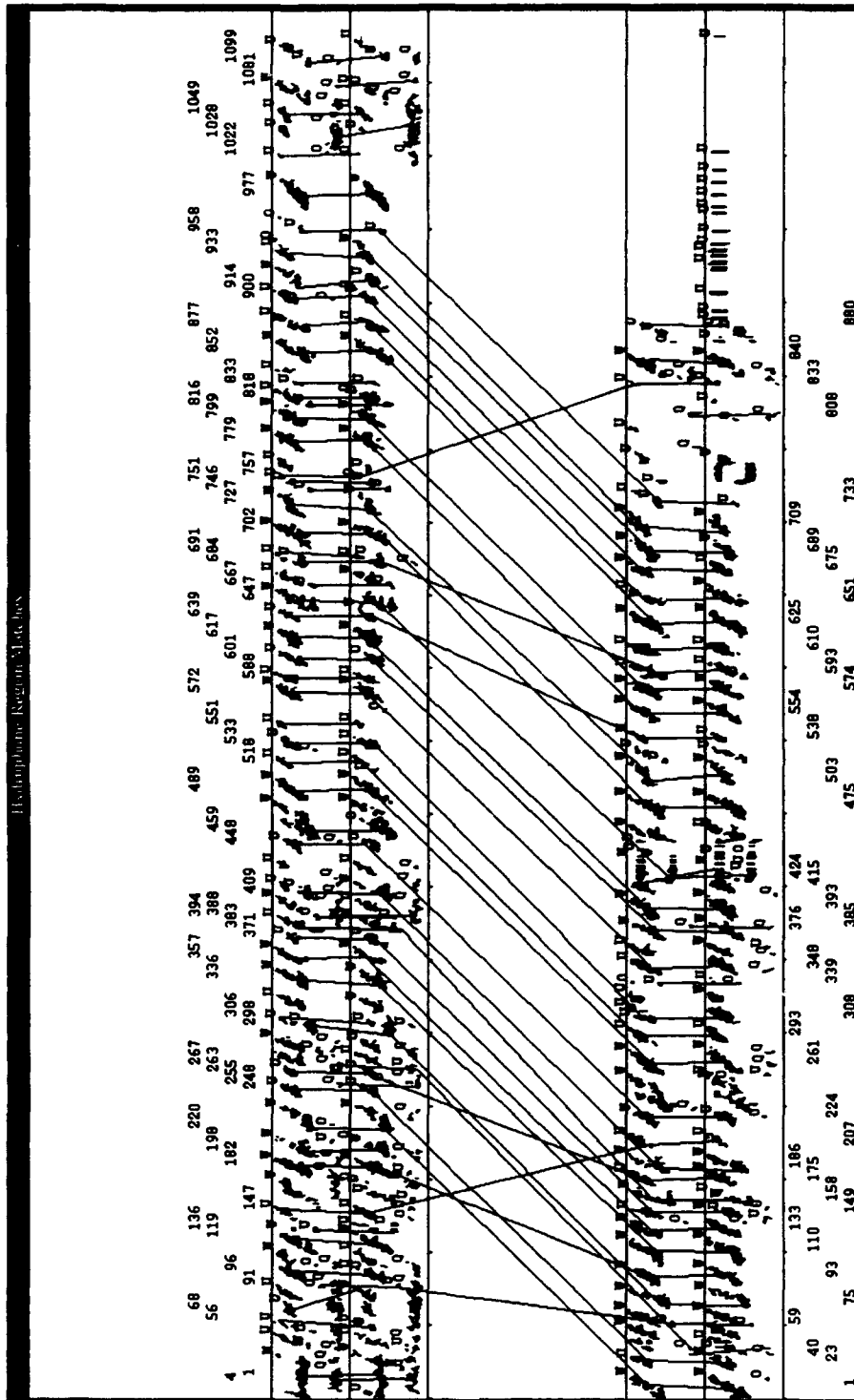














Hydrophone Region Matches

## APPENDIX C: PROLOG CODE FOR THE IDENTIFICATION PROGRAM

```

/* Incremental visual processing: operates line by line on a file. */
/* Note: to avoid counting a particular region, add an */
/* "uninteresting_region" fact for it--will help avoid space overflow. */

:- no_style_check(single_var), unknown(X,fail).
:- dynamic lastrowegensym/1, region_name_change/2, final_code/2,
   uninteresting_region/1, pixel/3, pixel/4.
/* SEEM SEEM SEEM SEEM SEEM SEEM SEEM SEEM SEEM SEEM SEEM SEEM SEEM SEEM SEEM SEEM */
:- dynamic lastwhalegensym/1,
   group_success/0, group/2, groupgensym/1, whale/3, outer_pixel/3.
:- asserta(average_lines(20)), asserta(threshold_inc(7)).
/* SEEM SEEM SEEM */

:- ensure_loaded(vision4).

demo :- demo(inputfile).
demo(Infile) :- incrsum(Infile,sumfile),
   newg(70), incrxgrow(Infile,growfile),
   incrshapes(finalgrowpict,shapesfile),
   tell(xfinalgrowpict), xshow(finalgrowpict), told,
   high_level.
rawdemo :- rawdemo(inputfile), do_lisp_regions.
rawdemo(Infile) :- newg(20), incrxgrow(Infile,growfile),
   incrshapes(finalgrowpict,shapesfile),
   tell(xfinalgrowpict), xshow(finalgrowpict), told,
   high_level.

/* Two ways to show a picture file: full data, or compressed data */
show(Infile) :- see(Infile), show2.
show2 :- not(at_end_of_file), readlineclump(L), printlist(L), nl, show2.
show2 :- seen.
xshow(Infile) :- see(Infile), xshow2.
xshow2 :- not(at_end_of_file), readlineclump(L), xprintlist(L), nl, xshow2.
xshow2 :- seen.
xprintlist([]).
xprintlist([N|L]) :- number(N), code62(N,A), put(A), !, xprintlist(L).
xprintlist([_|L]) :- write(I), !, xprintlist(L).
code62(N,A) :- M is (N mod 62), M<10, !, A is M+48.
code62(N,A) :- M is (N mod 62), M<36, !, A is M+55.
code62(N,A) :- M is (N mod 62), !, A is M+61.

incrsum(Infile,Outfile) :- incrmap4(Infile,Outfile,sum4).
incrgrad(Infile,Outfile) :- incrmap9(Infile,Outfile,gradientthreshold9).
incrgrow(Infile,Outfile) :-
   incrmap1(Infile,tempgrowfile,reverse_threshold),
   incrmap9(tempgrowfile,tempgrowfile2,deorphan),
   incrclump(tempgrowfile2,finalgrowpict),
   pixel_convert(finalgrowpict,temppixelfile),
   pixel_join(Infile,Outfile).

/* General-purpose line-by-line 2D array processing: applies the given */
/* function to every 3 by 3 square in the array, outputs array of results. */
/* Function must be the name of a 10-argument (9-input) function predicate. */
incrmap9(Infile,Outfile,Function) :- see(Infile), tell(Outfile),
   readlineclump(L1), readlineclump(L2), readlineclump(L3),
   getamap9(L1,L2,L3,Function), !.
getamap9(L1,L2,L3,Function) :- at_end_of_file,
   amapitem9(L1,L2,L3,Function,GL), write_line(GL),
   seen, told, !.
getamap9(L1,L2,L3,Function) :- amapitem9(L1,L2,L3,Function,GL),
   write_line(GL), readlineclump(L4), getamap9(L2,L3,L4,Function).

```



```

    retract(intensity(Line,N,G,I)), AvgI is I / N, IntAvgI is integer(AvgI),
    assertz(intensity(Line,N,G,IntAvgI)).
get_average_intensity(Line).

reverse_x_threshold(A,*) :- gradient_threshold(G), A<G,!.
reverse_x_threshold(A,0) :- last_line(Line), retract(intensity(Line,X,G,I)),!,
    NewI is I + A, Xp1 is X+1,!, assertz(intensity(Line,Xp1,G,NewI)),!.

/* Fill in gaps across rows, columns, and diagonals to make fuller shapes */
incrfill(Infile,Outfile) :- see(Infile), tell(Outfile), readlineclump(L1),
    readlineclump([L2a|L2]), readlineclump(L3), write_line(L1),
    incrfill2(L1,[L2a|L2],L3,L3,[L2a]), seen, told,!.
incrfill2(L1,L2,_,L3,L2new2) :- at_end_of_file, write_line(L2),
    write_line(L3),!.
incrfill2(L1,[_|L2b,L2c],L3,[L3a|L3orig],L2new1) :-
    append(L2new1,[L2b,L2c],L2new2), write_line(L2new2), readlineclump(L4),
    incrfill2(L2new2,[L3a|L3orig],L4,L4,[L3a]).
incrfill2([_|L1],[_,0|L2],[_|L3],L3orig,L2new1) :- append(L2new1,[0],L2new2),
    incrfill2(L1,[0|L2],L3,L3orig,L2new2).
incrfill2([_,L1b,0|L1],[_|L2],[0|L3],L3orig,L2new1) :-
    append(L2new1,[0],L2new2), incrfill2([L1b,0|L1],L2,L3,L3orig,L2new2).
incrfill2([0|L1],[_|L2],[_,L3b,0|L3],L3orig,L2new1) :-
    append(L2new1,[0],L2new2), incrfill2(L1,L2,[L3b,0|L3],L3orig,L2new2).
incrfill2([_,0|L1],[_|L2],[_,0|L3],L3orig,L2new1) :- append(L2new1,[0],L2new2),
    incrfill2([0|L1],L2,[0|L3],L3orig,L2new2).
incrfill2([_|L1],[0,_,0|L2],[_|L3],L3orig,L2new1) :- append(L2new1,[0],L2new2),
    incrfill2(L1,[0,0|L2],L3,L3orig,L2new2).
incrfill2([_|L1],[_,*|L2],[_|L3],L3orig,L2new1) :- append(L2new1,[*],L2new2),
    incrfill2(L1,[*|L2],L3,L3orig,L2new2).

/* Find man-made signals and mark them uniquely. */
get_lines(Infile,Outfile) :- see(Infile), tell(Outfile), readlineclump(L1),
    readlineclump(L2), readlineclump(L3), get_cal_signal(L2,L2s),
    get_cal_signal(L3,L3s), write_line(L1),
    get_lines2(L1,L2s,L3s,L3s,[]), seen, told,!.
get_lines2(L1,L2,L3old,L3,L2new) :- at_end_of_file, write_line(L2),
    write_line(L3),!.
get_lines2([*,m,m],[*,0,0],L3,L3orig,L2new1) :-
    append(L2new1,[*,m,m],L2new2), get_lone_pixels(L2new2,[],L2new3),
    write_line(L2new3), readlineclump(L4),
    get_cal_signal(L4,L4s), get_lines2(L2new2,L3orig,L4s,L4s,[]).
get_lines2(L1,[*,0,0],[*,0,0],L3orig,L2new1) :-
    append(L2new1,[*,m,m],L2new2), get_lone_pixels(L2new2,[],L2new3),
    write_line(L2new3), readlineclump(L4),
    get_cal_signal(L4,L4s), get_lines2(L2new2,L3orig,L4s,L4s,[]).
get_lines2(L1,[L2a,L2b,L2c],L3,L3orig,L2new1) :-
    append(L2new1,[L2a,L2b,L2c],L2new2), get_lone_pixels(L2new2,[],L2new3),
    write_line(L2new3), readlineclump(L4),
    get_cal_signal(L4,L4s), get_lines2(L2new2,L3orig,L4s,L4s,L3a).
get_lines2([*,*,*|L1],[*,0,0,*|L2],[*,*,*|L3],L3orig,L2new) :-
    append(L2new,[*,m,m],L2new2),
    get_lines2([*|L1],[*|L2],[*|L3],L3orig,L2new2).
get_lines2([m,*,*,*|L1],[*,0,0,*|L2],[*,*,*|L3],L3orig,L2new) :-
    append(L2new,[*,m,m],L2new2),
    get_lines2([*|L1],[*|L2],[*|L3],L3orig,L2new2).
get_lines2([*,*,*,m|L1],[*,0,0,*|L2],[*,*,*|L3],L3orig,L2new) :-
    append(L2new,[*,m,m],L2new2),
    get_lines2([*|L1],[*|L2],[*|L3],L3orig,L2new2).
get_lines2([*,0,0,*|L1],[*,0,0,*|L2],[_,_,_,L3d|L3],L3orig,L2new) :-
    append(L2new,[*,m,m],L2new2),
    get_lines2([*|L1],[*|L2],[L3d|L3],L3orig,L2new2).
get_lines2([*,m,m,*|L1],[*,0,0,*|L2],[_,_,_,L3d|L3],L3orig,L2new) :-
    append(L2new,[*,m,m],L2new2),
    get_lines2([*|L1],[*|L2],[L3d|L3],L3orig,L2new2).
get_lines2([_,_,_,L1d|L1],[*,0,0,*|L2],[*,0,0,*|L3],L3orig,L2new) :-
    append(L2new,[*,m,m],L2new2),
    get_lines2([L1d|L1],[*|L2],[*|L3],L3orig,L2new2).
get_lines2([_|L1],[L2a|L2],[_|L3],L3orig,L2new) :-

```



```

append(L2new, [L2a], L2new2), get_lines2(L1, L2, L3, L3orig, L2new2).

/* Find the characteristic calibration signal and change signals to match. */
get_cal_signal([*, A, *, *, *, *, *, *, *, *,
*, *, *, *, *, *, *, *, *, *,
*, *, *, *, B, C, 0, 0, D, *,
*, *, *, *, *, *, *, *, *,
*, *, L], Ls) :-
set_to_calsignal(
[*, A, *, *, *, *, *, *, *, *,
*, *, *, *, *, *, *, *, *,
*, *, *, *, B, C, 0, 0, D, *,
*, *, *, *, *, *, *, *, *,
*, *, L], [], Ls).
get_cal_signal([*, A, *, *, *, *, *, *, *, *,
*, *, *, *, *, *, *, *, *,
*, *, *, *, B, C, 0, D, E, *,
*, *, *, *, *, *, *, *, *,
*, *, L], Ls) :-
set_to_calsignal(
[*, A, *, *, *, *, *, *, *, *,
*, *, *, *, *, *, *, *, *,
*, *, *, *, B, C, 0, D, E, *,
*, *, *, *, *, *, *, *, *,
*, *, L], [], Ls).
get_cal_signal([*, A, *, *, *, *, *, *, *, *,
*, *, *, *, *, *, *, *, *,
*, *, *, *, B, C, D, 0, E, *,
*, *, *, *, *, *, *, *, *,
*, *, L], Ls) :-
set_to_calsignal(
[*, A, *, *, *, *, *, *, *, *,
*, *, *, *, *, *, *, *, *,
*, *, *, *, B, C, D, 0, E, *,
*, *, *, *, *, *, *, *, *,
*, *, L], [], Ls).
get_cal_signal([*, A, *, *, *, *, *, *, *, *,
*, *, *, *, *, *, *, *, *,
*, *, *, *, *, *, *, *, *,
*, *, *, *, *, *, *, *, *,
*, *, *, *, B, C, 0, 0, D, *, *, *, L], Ls) :-
set_to_calsignal(
[*, A, *, *, *, *, *, *, *, *,
*, *, *, *, *, *, *, *, *,
*, *, *, *, *, *, *, *, *,
*, *, *, *, *, *, *, *, *,
*, *, *, *, *, *, *, *, *,
*, *, *, *, B, C, 0, 0, D, *, *, *, L], [], Ls).
get_cal_signal(L, L).

set_to_calsignal([], Ls, Ls).
set_to_calsignal([*, L], Ls, Lsnew) :- append(Ls, [*, L], Lsnew2),
set_to_calsignal(L, Lsnew2, Lsnew).
set_to_calsignal([0, L], Ls, Lsnew) :- append(Ls, [c], Lsnew2),
set_to_calsignal(L, Lsnew2, Lsnew).

/* Get pixels that have no upper or lower neighbor and transform into an m */
/* These pixels will be put with the group of probable man-made pixels. */
get_lone_pixels([], Ls, Ls).
get_lone_pixels([*, 0], L, Ls) :- append(L, [*, m], Ls).
get_lone_pixels([*, 0, *|Rest], L, Ls) :- append(L, [*, m], Lnew),
get_lone_pixels([*, *|Rest], Lnew, Ls).
get_lone_pixels([L1|Rest], L, Ls) :- append(L, [L1], Lnew),
get_lone_pixels(Rest, Lnew, Ls).

/* Remove all the m and c markings from a file. */
remove_manmade_signals(Infile, Outfile) :- see(Infile), tell(Outfile),

```

```

readlineclump(L1), remove_manmade_signals2(L1,X), seen, told, !.
remove_manmade_signals2(L1,L1new) :- at_end_of_file, write_line(L1), !.
remove_manmade_signals2([m],L1) :- append(L1,[*],L1new), write_line(L1new),
readlineclump(L2), remove_manmade_signals2(L2,[]).
remove_manmade_signals2([X],L1) :- append(L1,[X],L1new), write_line(L1new),
readlineclump(L2), remove_manmade_signals2(L2,[]).
remove_manmade_signals2([m|L1],L1new) :- append(L1new,[*],L1newer),
remove_manmade_signals2(L1,L1newer).
remove_manmade_signals2([c|L1],L1new) :- append(L1new,[*],L1newer),
remove_manmade_signals2(L1,L1newer).
remove_manmade_signals2([X|L1],L1new) :- append(L1new,[X],L1newer),
remove_manmade_signals2(L1,L1newer).

/* Remove pixels that have no upper or lower neighbors. */
/* This will tend to get rid of any straight (man-made) lines in the data. */
remove_lone_pixels(Infile,Outfile) :- see(Infile), tell(Outfile),
readlineclump([X|L1]), remove_lone_pixels2([X|L1],[X]), seen, told, !.
remove_lone_pixels2(L1,L1new) :- at_end_of_file, write_line(L1), !.
remove_lone_pixels2([X1,X2],L1) :- append(L1,[X2],L1new), write_line(L1new),
readlineclump([X3|L2]), remove_lone_pixels2([X3|L2],[X3]).
remove_lone_pixels2([*,0,*|L1],L1new) :- append(L1new,[*],L1newer),
remove_lone_pixels2([*,*|L1],L1newer).
remove_lone_pixels2([X1,X2|L1],L1new) :- append(L1new,[X2],L1newer),
remove_lone_pixels2([X2|L1],L1newer).

get_pixel_values(Infile) :- see(Infile), readlineclump(L1),
get_line_values(1,L1),seen.
get_line_values(Line,List) :- at_end_of_file,
get_line_values2(1,Line,List).
get_line_values(Line,List) :- get_line_values2(1,Line,List),
NewLine is Line + 1, readlineclump(NewList),
get_line_values(NewLine,NewList).
get_line_values2(Row,Line,[]).
get_line_values2(Row,Line,[First|Rest]) :- pixel(Row,Line,X),
retract(pixel(Row,Line,X)), assert(pixel(Row,Line,X,First)),
NewRow is Row + 1, get_line_values2(NewRow,Line,Rest).
get_line_values2(Row,Line,[First|Rest]) :- NewRow is Row + 1,
get_line_values2(NewRow,Line,Rest).

/*
look_for_vertical_line :- last_line(Line), numzeros(Line,Zeros),
Linem1 is Line - 1, numzeros(Linem1,ZerosLm1), Dif is Zeros - ZerosLm1,
abs(Dif,AbsDif), AbsDif > 14, assertz(vertical_line(Line)).
look_for_vertical_line.
*/

/* Group nearby regions together. Find whales, calibration signals, ... */
high_level :- circumference, get_group, reorder_groups, characterize_groups(1),
get_horizontal_lines, get_vertices, get_whale, get_calibration_signals,
get_earthquakes, get_unknowns, get_unknown_group_qualities,
get_more_whales, print_results.

/* Find the outer pixels in each region and assert them. */
/* This keeps us from having to test every pixel in every region to see
if two regions are in the same group. */
circumference :- abolish(outer_pixel/3), circumference2(1).
circumference2(Region) :- do_circumference(Region), NewRegion is Region + 1,
area(NewRegion,X), circumference2(NewRegion).
circumference2(Region).
do_circumference(AreaNo) :- pixel(X,Y,AreaNo,_), not(inner_pixel(X,Y,AreaNo)),
assertz(outer_pixel(X,Y,AreaNo)), fail.
do_circumference(AreaNo).
inner_pixel(X,Y,AreaNo) :- pixel(X,Y,AreaNo,_),
left(X,Y,AreaNo), right(X,Y,AreaNo), up(X,Y,AreaNo), down(X,Y,AreaNo).
left(X,Y,AreaNo) :- LeftCell is X-1, pixel(LeftCell,Y,AreaNo,_).
right(X,Y,AreaNo) :- RightCell is X+1, pixel(RightCell,Y,AreaNo,_).
up(X,Y,AreaNo) :- UpCell is Y+1, pixel(X,UpCell,AreaNo,_).

```

```

down(X,Y,AreaNo) :- DownCell is Y-1, pixel(X,DownCell,AreaNo,_).

/* Group nearby regions together into groups of regions. */
get_group :- abolish(lastgroupgensym/1), abolish(group/2), get_group(1).
get_group(Region) :- get_group2(Region), NewRegion is Region + 1,
    area(NewRegion,X), get_group(NewRegion).
get_group(Region).
get_group2(R1) :- abolish(group_success/0), lastgroupgensym(GroupID),
    Gid is GroupID, get_group3(R1,GroupID,Gid),combine_regions(R1,GroupID).
get_group2(R1) :- groupgensym(K), assertz(group(K,[R1])), get_group4(R1,K),
    combine_regions(R1,K).
get_group3(R1,GroupID,Gid) :- GroupID < Gid + 3, get_group4(R1,Gid),
    group_success.
get_group3(R1,GroupID,Gid) :- GroupID < Gid + 3, NewGid is Gid - 1,
    get_group3(R1,GroupID, NewGid).
get_group4(R1,GroupID) :- group(GroupID,Glist), member(R1,Glist), R2 is ? 8,
    add_nearby_groups(R1,R2,GroupID), asserta(group_success).

/* If a region is near a group, then add it to that group. */
add_nearby_groups(R1,R2,GroupID) :- R2 > R1, group(GroupID,Glist),
    not(member(R2,Glist)), nearby(R1,R2), append([R2],Glist,NewGlist),
    retract(group(GroupID,Glist)),assertz(group(GroupID,NewGlist)),
    R3 is R2 - 1, add_nearby_groups(R1,R3,GroupID),!.
add_nearby_groups(R1,R2,GroupID) :- R2 > R1, R3 is R2 - 1,
    add_nearby_groups(R1,R3,GroupID),!.
add_nearby_groups(R1,R2,GroupID).

/* If a region is a member of two groups, then combine the two groups. */
combine_regions(R1,G1) :- G2 is G1 - 8, combine_regions2(R1,G1,G2).
combine_regions2(R1,G1,G2) :- G2 < G1, combine_groups(R1,G1,G2), G3 is G2 + 1,
    combine_regions2(R1,G1,G3).
combine_regions2(R1,G1,G2).
combine_groups(R1,G1,G2) :- G1 \= G2, group(G1,G1list), member(R1,G1list),
    group(G2,G2list), member(R1,G2list), union(G1list,G2list,G3list),
    retract(group(G1,G1list)), retract(group(G2,G2list)),
    assertz(group(G1,G3list)).
combine_groups(R1,G1,G2).

/* Find out if two regions are close enough together to be called nearby. */
nearby(R1,R2) :- outer_pixel(X1,Y1,R1), outer_pixel(X2,Y2,R2),
    Xdif is X1 - X2, Ydif is Y1 - Y2, Dist is Xdif * Xdif + Ydif * Ydif,
    Dist < 13.

/* Due to group combining, there are missing numbers in the groups. */
/* Need to have a straight sequence of groups for later processing. */
reorder_groups :- lastgroupgensym(Gmax), get_start_groups(G1,G2,Gmax),
    reorder_groups2(G1,G2,Gmax).
reorder_groups2(G1,G2,Gmax) :- G2 < Gmax + 1, group(G2,G2l), G1p1 is G1 + 1,
    retract(group(G2,G2l)), assertz(group(G1p1,G2l)), G2p1 is G2 + 1,
    get_nonempty_group(G2p1,G3,Gmax), reorder_groups2(G1p1,G3,Gmax).
reorder_groups2(G1,G2,Gmax).

/* Return G1 as the first empty group, and G2 as the first non-empty group */
get_start_groups(G1,G2,Gmax) :- get_empty_group(1,G1p1), G1 is G1p1 - 1,
    G1p2 is G1p1 + 1, get_nonempty_group(G1p2,G2,Gmax).

/* Return next valid group number */
get_nonempty_group(G1,G1,Gmax) :- group(G1,G1l).
get_nonempty_group(G1,G1,Gmax) :- G1 > Gmax.
get_nonempty_group(G1,G2,Gmax) :- G3 is G1 + 1, get_nonempty_group(G3,G2,Gmax).

/* Return next invalid group number */
get_empty_group(G1,G1) :- not(group(G1,G1l)).
get_empty_group(G1,G2) :- G3 is G1 + 1, get_empty_group(G3,G2).

/* Find out the basic parameters for each group and assert them. */
characterize_groups(Group) :- group(Group,G1), characterize_groups2(Group),
    NewGroup is Group + 1, characterize_groups(NewGroup).

```

```

characterize_groups(Group).
character_groups2(Group) :- group(Group,Glist),
    character_groups3(Glist,Xmin,Xmax,Ymin,Ymax,Area,IntDPct,IntAvgI),
    assertz(group_qualities(Group,Xmin,Xmax,Ymin,Ymax,Area,IntAvgI,IntDPct)).
character_groups3(Glist,Xmin,Xmax,Ymin,Ymax,Area,IntDPct,IntAvgI) :-
    get_bounding_box(Glist,Xmin,Xmax,Ymin,Ymax), get_area(Glist,Area),
    get_density(Xmin,Xmax,Ymin,Ymax,Area,Density),
    IntDPct is integer(Density),
    get_total_intensity(Glist,Loudness), AvgI is Loudness / Area,
    IntAvgI is integer(AvgI).
get_bounding_box(Glist,Xmin,Xmax,Ymin,Ymax) :- get_xmin(Glist,Xmin),
    get_xmax(Glist,Xmax), get_ymin(Glist,Ymin), get_ymax(Glist,Ymax).
get_xmin(Glist,Xmin) :- get_xmin2(Glist,99999,Xmin).
get_xmin2([],Xmin,Xmin).
get_xmin2([G|List],Tempmin,Xmin) :- xmin(G,Min), Min < Tempmin,
    get_xmin2(List,Min,Xmin).
get_xmin2([G|List],Tempmin,Xmin) :- get_xmin2(List,Tempmin,Xmin).
get_xmax(Glist,Xmax) :- get_xmax2(Glist,0,Xmax).
get_xmax2([],Xmax,Xmax).
get_xmax2([G|List],Tempmax,Xmax) :- xmax(G,Max), Max > Tempmax,
    get_xmax2(List,Max,Xmax).
get_xmax2([G|List],Tempmax,Xmax) :- get_xmax2(List,Tempmax,Xmax).
get_ymin(Glist,Ymin) :- get_ymin2(Glist,99999,Ymin).
get_ymin2([],Ymin,Ymin).
get_ymin2([G|List],Tempmin,Ymin) :- ymin(G,Min), Min < Tempmin,
    get_ymin2(List,Min,Ymin).
get_ymin2([G|List],Tempmin,Ymin) :- get_ymin2(List,Tempmin,Ymin).
get_ymax(Glist,Ymax) :- get_ymax2(Glist,0,Ymax).
get_ymax2([],Ymax,Ymax).
get_ymax2([G|List],Tempmax,Ymax) :- ymax(G,Max), Max > Tempmax,
    get_ymax2(List,Max,Ymax).
get_ymax2([G|List],Tempmax,Ymax) :- get_ymax2(List,Tempmax,Ymax).
get_area(Glist,Area) :- get_area2(Glist,0,Area).
get_area2([],Area,Area).
get_area2([G|List],OldArea,Area) :- area(G,PartArea),
    NewArea is OldArea + PartArea, get_area2(List,NewArea,Area).
get_density(Xmin,Xmax,Ymin,Ymax,Area,Density) :-
    Xdif is Xmax - Xmin + 1, Ydif is Ymax - Ymin + 1, BoxArea is Xdif * Ydif,
    Density is 100.0 * Area / BoxArea.
get_total_intensity(Glist,Loudness) :- abolish(loudness/1),
    asserta(loudness(0)), get_total_intensity2(Glist),loudness(Loudness).
get_total_intensity2([]).
get_total_intensity2([R|L]) :- pixel(X,Y,R,Loud), retract(loudness(Loudness)),
    NewLoud is Loud + Loudness, asserta(loudness(NewLoud)), fail.
get_total_intensity2([R|L]) :- get_total_intensity2(L).

/* Find all of the horizontal lines in the data set */
get_horizontal_lines :- abolish(horizontal_line/4), get_horizontal_lines(1).
get_horizontal_lines(Group) :- group(Group,Glist),
    add_group_field_to_outer_pixels(Group,Glist),
    get_horizontal_lines2(Group),
    NewGroup is Group + 1, get_horizontal_lines(NewGroup).
get_horizontal_lines2(Group) :-
    bagof((X,Ylist), bagof(Y,R^outer_pixel(X,Y,R,Group),Ylist), XYlist),
    get_horizontal_lines3(Group,XYlist).
get_horizontal_lines2(Group,List).
get_horizontal_lines3(Group,[]).
get_horizontal_lines3(Group,[(X,Ylist)|XYlist]) :- length(Ylist,Ylen),
    Ylen > 8, min(Ylist,Ymin), max(Ylist,Ymax), Ydif is Ymax - Ymin + 1,
    Density is Ylen / Ydif, Density > 0.5,
    assertz(horizontal_line(Group,X,Ymin,Ymax)),
    get_horizontal_lines3(Group,XYlist).
get_horizontal_lines3(Group,[(X,Ylist)|XYlist]) :-
    get_horizontal_lines3(Group,XYlist).

/* Take the 3 argument outer_pixel facts and associate a group with them */
add_group_field_to_outer_pixels(Group,[]).

```

```

add_group_field_to_outer_pixels(Group, {Region|Glist}) :-
    change_outer_pixels(Group, Region),
    add_group_field_to_outer_pixels(Group, Glist).
change_outer_pixels(Group, Region) :- outer_pixel(X, Y, Region),
    retract(outer_pixel(X, Y, Region)), assertz(outer_pixel(X, Y, Region, Group)),
    fail.
change_outer_pixels(Group, Region).

/* Find intersections between horizontal and vertical lines and assert vertex.*/
get_vertices :- get_vertices1, remove_duplicate_vertices.
get_vertices1 :- vertical_line(Y), get_vertices2(Y), retract(vertical_line(Y)),
    get_vertical_edge_regions(Y, Rlist), assertz(vertical_edge(Y, Rlist)), fail.
get_vertices1.
get_vertices2(Y) :- horizontal_line(Group, X, Ymin, Ymax), Ydif is Ymax - Y,
    abs(Ydif, AbsYdif), AbsYdif < 5, group(Group, Glist),
    not(vertex(Group, Glist, X, Y)), assertz(vertex(Group, Glist, X, Y)), fail.
get_vertices2(Y) :- horizontal_line(Group, X, Ymin, Ymax), Ydif is Ymin - Y,
    abs(Ydif, AbsYdif), AbsYdif < 5, group(Group, Glist),
    not(vertex(Group, Glist, X, Y)), assertz(vertex(Group, Glist, X, Y)), fail.
get_vertices2(Y) :- horizontal_line(Group, X, Ymin, Ymax), Y >= Ymin, Y <= Ymax,
    group(Group, Glist), not(vertex(Group, Glist, X, Y)),
    assertz(vertex(Group, Glist, X, Y)), fail.
get_vertices2(Y).

/* Take the vertical edge and associate it with an appropriate region */
get_vertical_edge_regions(Y, Rlist) :-
    setof(R, Group^X^outer_pixel(X, Y, R, Group), Rlist).

/* There may be several Y intercepts for the X lines. Just take nearest. */
remove_duplicate_vertices :- vertex(Group, Glist, X, Y1), vertex(Group, Glist, X, Y2),
    Y2 =\= Y1, horizontal_line(Group, X, Ymin, Ymax), Ydif1 is Ymax - Y1,
    abs(Ydif1, AbsYdif1), AbsYdif1 < 5, Ydif2 is Ymax - Y2, abs(Ydif2, AbsYdif2),
    AbsYdif1 < AbsYdif2, retract(vertex(Group, Glist, X, Y2)), fail.
remove_duplicate_vertices :- vertex(Group, Glist, X, Y1), vertex(Group, Glist, X, Y2),
    Y2 =\= Y1, horizontal_line(Group, X, Ymin, Ymax), Ydif1 is Ymax - Y1,
    abs(Ydif1, AbsYdif1), AbsYdif1 < 5, Ydif2 is Ymax - Y2, abs(Ydif2, AbsYdif2),
    AbsYdif1 >= AbsYdif2, retract(vertex(Group, Glist, X, Y1)), fail.
remove_duplicate_vertices :- vertex(Group, Glist, X, Y1), vertex(Group, Glist, X, Y2),
    Y2 =\= Y1, horizontal_line(Group, X, Ymin, Ymax), Ydif1 is Ymin - Y1,
    abs(Ydif1, AbsYdif1), AbsYdif1 < 5, Ydif2 is Ymin - Y2, abs(Ydif2, AbsYdif2),
    AbsYdif1 < AbsYdif2, retract(vertex(Group, Glist, X, Y2)), fail.
remove_duplicate_vertices :- vertex(Group, Glist, X, Y1), vertex(Group, Glist, X, Y2),
    Y2 =\= Y1, horizontal_line(Group, X, Ymin, Ymax), Ydif1 is Ymin - Y1,
    abs(Ydif1, AbsYdif1), AbsYdif1 < 5, Ydif2 is Ymax - Y2, abs(Ydif2, AbsYdif2),
    AbsYdif1 >= AbsYdif2, retract(vertex(Group, Glist, X, Y1)), fail.
remove_duplicate_vertices.

/* Find all the whale sounds in the data set. */
get_whale :- abolish(lastwhalegensym/1), abolish(whale/3), get_whale(1).
get_whale(Group) :- group(Group, Glist), get_whale2(Group),
    NewGroup is Group + 1, get_whale(NewGroup).
get_whale(Group).
get_whale2(Group) :-
    group_qualities(Group, Xmin, Xmax, Ymin, Ymax, Area, Loudness, Density),
    Xmin > 20, Ylength is Ymax - Ymin, Ylength > 12, Ylength < 60, Area > 50,
    Area < 500, Density > 8, Density < 75, whalegensym(K),
    group(Group, Glist), assertz(whale(K, Group, Glist)).
get_whale2(Group).

/* Find the calibration signals in the data set. */
/* Calibration signals are listed by time (YMIN, YMAX), instead of by regions.*/
get_calibration_signals :- abolish(low_calibration_signal/1),
    abolish(overtone_region/3), abolish(overtone_pixel/4),
    abolish(calibration_signal/4), get_low_calibration, get_overtone.

/* The calibration signals are found by keying on the lowest tone. */
get_low_calibration :-
    group_qualities(Group, Xmin, Xmax, Ymin, Ymax, Area, Loudness, Density),

```

```

Area > 10, Xmax < 4, group(Group,Glist),
assertz(low_calibration_signal(Group,Glist)), fail.
get_low_calibration.

/* The overtone signals are found after the low signal is identified. */
get_overtone :- low_calibration_signal(LowGroup,Glist),
get_overtone_regions(LowGroup), get_overtone_groups(LowGroup),
eliminate_extraneous_pixels(LowGroup), get_calibration_params(LowGroup),
fail.
get_overtone.
get_overtone_regions(LowGroup) :-
group_qualities(LowGroup,Lxmin,Lxmax,Lymin,Lymax,Larea,Lloudness,Ldensity),
ymin(R,Rymin), Rymin > Lymin, Rymin < Lymax,
group(LowGroup,Glist), not(overtone_region(LowGroup,Glist,R)),
assertz(overtone_region(LowGroup,Glist,R)), fail.
get_overtone_regions(LowGroup) :-
group_qualities(LowGroup,Lxmin,Lxmax,Lymin,Lymax,Larea,Lloudness,Ldensity),
ymax(R,Rymax), Rymax > Lymin, Rymax < Lymax,
group(LowGroup,Glist), not(overtone_region(LowGroup,Glist,R)),
assertz(overtone_region(LowGroup,Glist,R)), fail.
get_overtone_regions(LowGroup) :-
group_qualities(LowGroup,Lxmin,Lxmax,Lymin,Lymax,Larea,Lloudness,Ldensity),
ymax(R,Rymax), Rymax > Lymax, ymin(R,Rymin), Rymin < Lymin,
group(LowGroup,Glist), not(overtone_region(LowGroup,Glist,R)),
assertz(overtone_region(LowGroup,Glist,R)), fail.
get_overtone_regions(LowGroup) :-
group_qualities(LowGroup,Lxmin,Lxmax,Lymin,Lymax,Larea,Lloudness,Ldensity),
ymax(R,Rymax), Rymax > Lymax, ymin(R,Rymin), Rymin < Lymax,
group(LowGroup,Glist), not(overtone_region(LowGroup,R)),
assertz(overtone_region(LowGroup,Glist,R)), fail.
get_overtone_regions(LowGroup).
get_overtone_groups(LowGroup) :- overtone_region(LowGroup,LGlist,Region),
group(G,Glist), member(Region,Glist),
not(overtone_group(LowGroup,LGlist,G,Glist)),
assertz(overtone_group(LowGroup,LGlist,G,Glist)), fail.
get_overtone_groups(LowGroup).

/* Sometimes regions extend beyond the calibration signal. Pixels from these
regions must be eliminated in order to get only the calibration signal. */
eliminate_extraneous_pixels(LowGroup) :-
overtone_group(LowGroup,LGlist,OverGroup,OGList),
cull_region_pixels(LowGroup,OGList), fail.
eliminate_extraneous_pixels(LowGroup).
cull_region_pixels(G,[]).
cull_region_pixels(G,[R|List]) :- cull_region_pixels2(G,R),
cull_region_pixels(G,List).
cull_region_pixels2(G,R) :- pixel(X,Y,R,I), X > 26, X < 29,
assertz(overtone_pixel(X,Y,G,I)), fail.
cull_region_pixels2(G,R) :- pixel(X,Y,R,I), X > 50, X < 56,
assertz(overtone_pixel(X,Y,G,I)), fail.
cull_region_pixels2(G,R).

/* Find out when the calibration signal begins and ends. */
get_calibration_params(Group) :- get_overtone_params(Group,Oymin,Oymax),
get_calibration_ymin(Group,Oymin,Ymin),
get_calibration_ymax(Group,Oymax,Ymax),
group(Group,Glist), assertz(calibration_signal(Group,Glist,Ymin,Ymax)).
get_overtone_params(Group,Ymin,Ymax) :-
bagof(Y, I^X^overtone_pixel(X,Y,Group,I), Ylist),
min(Ylist,Ymin), max(Ylist,Ymax).
get_calibration_ymin(Group,Oymin,Ymin) :-
group_qualities(Group,Lxmin,Lxmax,Lymin,Lymax,Larea,Lloudness,Ldensity),
Lymin < Oymin, Ymin is Lymin.
get_calibration_ymin(Group,Oymin,Ymin) :- Ymin is Oymin.
get_calibration_ymax(Group,Oymax,Ymax) :-
group_qualities(Group,Lxmin,Lxmax,Lymin,Lymax,Larea,Lloudness,Ldensity),
Lymax > Oymax, Ymax is Lymax.
get_calibration_ymax(Group,Oymax,Ymax) :- Ymax is Oymax.

```

```

/* Process all groups to see if they might be earthquakes */
get_earthquakes :- abolish(lastquakegensym/1), abolish(lastfarquakegensym/1),
abolish(possible_quake/3), abolish(far_quake/2),
get_potential_quakes(1), get_far_quakes(1), define_far_quakes(1),
get_near_quakes.
get_potential_quakes(Group) :- group(Group,Glist), get_potential_quakes2(Group),
NewGroup is Group + 1, get_potential_quakes(NewGroup).
get_potential_quakes(Group).
get_potential_quakes2(Group) :-
group_qualities(Group,Xmin,Xmax,Ymin,Ymax,Area,Loudness,Density),
Xmin < 10, Area > 5,
not(calibration_signal(Group,_,_,_)), group(Group,Glist), quakegensym(K),
assertz(possible_quake(K,Group,Glist)).
get_potential_quakes2(Group).

get_far_quakes(Quake) :- possible_quake(Quake,Group,Glist),
group_qualities(Group,Xmin,Xmax,Ymin,Ymax,Area,Loudness,Density), Xmax < 25,
get_far_quakes2(Quake), NewQuake is Quake + 1, get_far_quakes(NewQuake).
get_far_quakes(Quake) :- possible_quake(Quake,Group,Glist),
NewQuake is Quake + 1, get_far_quakes(NewQuake).
get_far_quakes(Quake).
get_far_quakes2(Quake) :- combine_adjacent_far_quakes(Quake).
get_far_quakes2(Quake) :- farquakegensym(K), assertz(far_quake(K,[Quake])).
get_near_quakes.

combine_adjacent_far_quakes(Quake) :- lastfarquakegensym(K), far_quake(K,Qlist),
Quakem1 is Quake - 1, possible_quake(Quakem1,Groupm1,_),
member(Quakem1,Qlist), possible_quake(Quake,Group,_),
group_qualities(Group,_,_,Ymin,Ymax,_,_,_),
group_qualities(Groupm1,_,_,Ymaxm1,_,_,_), !,
adjacent_quakes(Ymin,Ymax,Ymaxm1),
retract(far_quake(K,Qlist)), assertz(far_quake(K,[Quake|Qlist])).
adjacent_quakes(Ymin,Ymax,Ymaxm1) :-
Ydif is Ymaxm1 - Ymin, abs(Ydif,AbsYdif), AbsYdif < 9.
adjacent_quakes(Ymin,Ymax,Ymaxm1) :- Ymaxm1 > Ymax.

define_far_quakes(Quake) :- far_quake(Quake,Qlist),
get_quake_regionlist(Qlist,[],Rlist), retract(far_quake(Quake,Qlist)),
assertz(far_quake(Quake,Rlist)), NewQuake is Quake + 1,
define_far_quakes(NewQuake).
define_far_quakes(Quake).
get_quake_regionlist([],Rlist,Rlist).
get_quake_regionlist([Quake|Qlist],[],Rlist) :-
possible_quake(Quake,Group,Glist), get_quake_regionlist(Qlist,Glist,Rlist).
get_quake_regionlist([Quake|Qlist],Reglist,Rlist) :-
possible_quake(Quake,Group,Glist),
append(Glist,Reglist,Newlist), get_quake_regionlist(Qlist,Newlist,Rlist).

/* Assert facts for all groups that haven't been included in other types */
get_unknowns :- abolish(lastunknowngensym/1), abolish(unknown_group/3),
get_unknowns1(1).
get_unknowns1(Group) :- group(Group,Glist),
group_qualities(Group,_,_,_,Area,_,_), Area > 7, not(whale_member(Group)),
not(possible_quake_member(Group)), not(overtone_group_member(Group)),
unknowngensym(K), assertz(unknown_group(K,Group,Glist)),
NewGroup is Group + 1, get_unknowns1(NewGroup).
get_unknowns1(Group) :- group(Group,Glist), NewGroup is Group + 1,
get_unknowns1(NewGroup).
get_unknowns1(Group).

whale_member(Group) :- whale(_,Group,_).
possible_quake_member(Group) :- possible_quake(_,Group,_).
overtone_group_member(Group) :- overtone_group(_,_,Group,_).
overtone_group_member(Group) :- overtone_group(Group,_,_,_).

/* Find the angle of the best line fit through a group */

```

```

/* Also find the mean distance and variation of the pixels in the group from */
/* the line. Also, find the mean distance and variation of the intersection */
/* of the perpendicular to each pixel with the line. */
get_unknown_group_qualities :- abolish(best_line_fit/6),
get_uk_group_qualities(1).
get_uk_group_qualities(Ugroup) :- unknown_group(Ugroup,Group,Glist),
get_uk_group_qualities2(Ugroup), NewGroup is Ugroup + 1,
get_uk_group_qualities(NewGroup).
get_uk_group_qualities(Ugroup).
get_uk_group_qualities2(Group) :- get_center(Group,Xc,Yc),
assertz(best_line_fit(Group,10,10,10,10,10)), abolish(xdist/1),
abolish(xdistvariance/1), abolish(dist_to_line/1),
abolish(dist_to_line_variance/1), assertz(xdist(0)), assertz(xdistvariance(0)),
assertz(dist_to_line(0)), assertz(dist_to_line_variance(0)),
get_uk_group_qualities3(Group,Xc,Yc,0.0),
best_line_fit(Group,Angle,_,_,_).
get_uk_group_qualities3(Group,Xc,Yc,Angle) :- Angle > 2.75.
get_uk_group_qualities3(Group,Xc,Yc,Angle) :-
get_line_eq(Xc,Yc,Angle,SinBeta,CosBeta,P),group(Group,Glist),
get_distances(Glist,Xc,Yc,SinBeta,CosBeta,P),get_distances2(Group),
get_best_line_fit(Group,Angle), NewAngle is Angle + 0.392699,
get_uk_group_qualities3(Group,Xc,Yc,NewAngle).
get_distances([],Xc,Yc,SinBeta,CosBeta,P).
get_distances([Region|Group],Xc,Yc,SinBeta,CosBeta,P) :- pixel(X,Y,Region,_),
DistToLine is X * CosBeta + Y * SinBeta - P, abs(DistToLine,AbsDistToLine),
DistToLineSQ is DistToLine * DistToLine, Xdif is Xc - X, Ydif is Yc - Y,
DistToCenterSQ is Xdif * Xdif + Ydif * Ydif,
XdistSQ is DistToCenterSQ - DistToLineSQ, do_sqrt(XdistSQ,Xdist),
retract(xdist(OldXdist)), NewXdist is OldXdist + Xdist,
assertz(xdist(NewXdist)),
retract(xdistvariance(OldXdistSQ)), NewXdistSQ is OldXdistSQ + XdistSQ,
assertz(xdistvariance(NewXdistSQ)),
retract(dist_to_line(OldDistToLine)),
NewDistToLine is OldDistToLine + AbsDistToLine,
assertz(dist_to_line(NewDistToLine)),
retract(dist_to_line_variance(OldDistToLineSQ)),
NewDistToLineSQ is OldDistToLineSQ + DistToLineSQ,
assertz(dist_to_line_variance(NewDistToLineSQ)),
fail.
get_distances([Region|Group],Xc,Yc,SinBeta,CosBeta,P) :-
get_distances(Group,Xc,Yc,SinBeta,CosBeta,P).
get_distances2(Group) :- group_qualities(Group,_,_,_,Area,_,_),
retract(xdist(OldXdist)), NewXdist is OldXdist / Area,
assertz(xdist(NewXdist)),
retract(xdistvariance(OldXdistVar)), NewXdistVar is OldXdistVar / Area,
assertz(xdistvariance(NewXdistVar)),
retract(dist_to_line(OldDistToLine)), NewDistToLine is OldDistToLine / Area,
assertz(dist_to_line(NewDistToLine)),
retract(dist_to_line_variance(OldDTLV)), NewDTLV is OldDTLV / Area,
assertz(dist_to_line_variance(NewDTLV)).
get_best_line_fit(Group,NewAngle) :-
best_line_fit(Group,Angle,Xdist,XdistVar,DistToLine,DistToLineVar),
dist_to_line(NewDTL), NewDTL < DistToLine,
xdist(NewXdist), xdistvariance(NewXDV), dist_to_line_variance(NewDTLV),
retract(best_line_fit(Group,Angle,Xdist,XdistVar,DistToLine,DistToLineVar)),
assertz(best_line_fit(Group,NewAngle,NewXdist,NewXDV,NewDTL,NewDTLV)).
get_best_line_fit(Group,NewAngle).
get_line_eq(Xc,Yc,Angle,SinBeta,CosBeta,P) :- acos(0,PiOver2),
Beta is Angle + PiOver2, sin(Beta,SinBeta), cos(Beta,CosBeta),
P is Xc * CosBeta + Yc * SinBeta.
do_sqrt(XdistSQ,Xdist) :- XdistSQ > 0, sqrt(XdistSQ,Xdist),!.
do_sqrt(XdistSQ,Xdist) :- Xdist is 0.
get_center(Group,Xc,Yc) :- group(Group,Glist), abolish(xtotal/1),
abolish(ytotal/1), assertz(xtotal(0)),assertz(ytotal(0)),
get_xy_totals(Glist), group_qualities(Group,_,_,_,Area,_,_),
xtotal(Xt), ytotal(Yt), Xc is Xt / Area, Yc is Yt / Area.
get_xy_totals([]).
get_xy_totals([Region|Glist]) :- pixel(X,Y,Region,_),xtotal(Xt),ytotal(Yt),

```



```

NewXt is Xt + X, NewYt is Yt + Y, retract(xtotal(Xt)), retract(ytotal(Yt)),
assertz(xtotal(NewXt)), assertz(ytotal(NewYt)), fail.
get_xy_totals([Region|Glist]) :- get_xy_totals(Glist).

/* Take unknown groups and find out if they might be associated with whales. */
get_more_whales :- get_more_whales1(1).
get_more_whales1(Wgroup) :- whale(Wgroup,Group,_), get_more_whales2(Group),
    NewWgroup is Wgroup + 1, get_more_whales1(NewWgroup).
get_more_whales1(Wgroup).
get_more_whales2(Wgroup) :-
    group_qualities(Wgroup,WXmin,WXmax,WYmin,WYmax,WArea,_,_),
    unknown_group(Ugroup,Group,Glist),
    group_qualities(Group,UXmin,UXmax,UYmin,UYmax,UArea,_,_),
    nearby_whale(UXmin,WXmin,UXmax,WXmax,UYmin,WYmin,UYmax,WYmax),
    retract(unknown_group(Ugroup,Group,Glist)),
    assert(associated_whale_group(Ugroup,Group,Glist)),
    make_whale_group_list(Wgroup,Group).
get_more_whales2(Wgroup).

/* Determine if an unknown sound is near enough to a whale sound to be called
a part of the same sound. */
nearby_whale(UXmin,WXmin,UXmax,WXmax,UYmin,WYmin,UYmax,WYmax) :-
    XminMaxDif is UXmin - WXmax, abs(XminMaxDif,AbsXminMaxDif),
    AbsXminMaxDif < 14, YminMaxDif is UYmin - WYmax,
    abs(YminMaxDif,AbsYminMaxDif), AbsYminMaxDif < 8.
nearby_whale(UXmin,WXmin,UXmax,WXmax,UYmin,WYmin,UYmax,WYmax) :-
    XminMaxDif is WXmin - UXmax, abs(XminMaxDif,AbsXminMaxDif),
    AbsXminMaxDif < 14, YminMaxDif is WYmin - UYmax,
    abs(YminMaxDif,AbsYminMaxDif), AbsYminMaxDif < 8.

/* Add to list of grouped groups of whales or create a new one */
make_whale_group_list(Wgroup,Group) :- whale_group(WhaleGroup),
    member(Wgroup,WhaleGroup), retract(whale_group(WhaleGroup)),
    append([Group],WhaleGroup,NewWhaleGroup),
    assert(whale_group(NewWhaleGroup)).
make_whale_group_list(Wgroup,Group) :- assert(whale_group([Wgroup,Group])).

/*
get_more_whales1(Ugroup) :- unknown_group(Ugroup,Group,Glist),
    group_qualities(Ugroup,Xmin,_,Ymin,Ymax,Area,_,_), Xmin > 10,
    Ymax - Ymin > 4, Area > 20, Area < 240, best_line_fit(Ugroup,Angle,_,_,_,_),
    get_more_whales2(Ugroup,Group,Angle), NewUgroup is Ugroup + 1,
    get_more_whales1(NewUgroup).
get_more_whales1(Ugroup) :- unknown_group(Ugroup,Group,Glist),
    NewUgroup is Ugroup + 1, get_more_whales1(NewUgroup).
get_more_whales1(Ugroup).
get_more_whales2(Ugroup,Group,Angle) :- Angle > 0.30, Angle < 1.2,
    whalegensym(K), group(Group,Glist), assertz(whale(K,Group,Glist)),
    retract(unknown_group(Ugroup,Group,Glist)),
    assertz(unknown_group(Ugroup,0,[0])).
get_more_whales2(Ugroup,Group,Angle).
*/

/*
get_overlapping_whales :- get_overlapping_whales1(1).
get_overlapping_whales1(Ugroup) :- unknown_group(Ugroup,Group,Glist),
    Group > 0, group_qualities(Ugroup,Xmin,_,Ymin,Ymax,Area,_,_),
    Xmin > 10, Ymax - Ymin > 50, Area > 239, get_overlapping_whales2(Ugroup),
    NewUgroup is Ugroup + 1, get_overlapping_whales1(NewUgroup).
get_overlapping_whales1(Ugroup) :- unknown_group(Ugroup,Group,Glist),
    NewUgroup is Ugroup + 1, get_more_whales1(NewUgroup).
get_overlapping_whales1(Ugroup).
get_overlapping_whales2(Ugroup) :- group(Ugroup,Glist), member(R,Glist),
    delete(R,Glist,NewGlist),
*/

```

```

/* Generates a new integer each time it is called */
whalegensym(K) :- retract(lastwhalegensym(Km1)), !, K is Km1+1,
    assertz(lastwhalegensym(K)).
whalegensym(1) :- assertz(lastwhalegensym(1)).

groupgensym(K) :- retract(lastgroupgensym(Km1)), !, K is Km1+1,
    assertz(lastgroupgensym(K)).
groupgensym(1) :- assertz(lastgroupgensym(1)).

quakegensym(K) :- retract(lastquakegensym(Km1)), !, K is Km1+1,
    assertz(lastquakegensym(K)).
quakegensym(1) :- assertz(lastquakegensym(1)).

farquakegensym(K) :- retract(lastfarquakegensym(Km1)), !, K is Km1+1,
    assertz(lastfarquakegensym(K)).
farquakegensym(1) :- assertz(lastfarquakegensym(1)).

unknowngensym(K) :- retract(lastunknowngensym(Km1)), !, K is Km1+1,
    assertz(lastunknowngensym(K)).
unknowngensym(1) :- assertz(lastunknowngensym(1)).

print_results :- tell(results), listing(group), listing(group_qualities),
    listing(unknown_group), listing(best_line_fit), listing(horizontal_line),
    listing(vertical_edge), listing(vertex), listing(calibration_signal),
    listing(overtone_region), listing(low_calibration_signal),
    listing(overtone_group), listing(whale), listing(possible_quake),
    listing(far_quake), listing(numpixels), told.

/* Write the regions of size > 5 to a file in lisp readable format. */
do_lisp_regions :- tell(lispfile), write_pixel_group(1), told.
write_pixel_group(G) :- group(G,Glist), already_written(G), Gp1 is G + 1,
    write_pixel_group(Gp1).
write_pixel_group(G) :- group(G,Glist), group_qualities(G,_,_,_,Area,_,_),
    Area > 5, write('('), group_type(G,Type,NewGlist), write(Type), write(' '),
    write_pixel_group2(NewGlist),
    write(')'), Gp1 is G + 1, write_pixel_group(Gp1).
write_pixel_group(G) :- group(G,Glist), Gp1 is G + 1, write_pixel_group(Gp1).
write_pixel_group(G).
write_pixel_group2({}).
write_pixel_group2([Region|Glist]) :- write_pixel_group3(Region),
    write_pixel_group2(Glist).
write_pixel_group3(Region) :- pixel(Y,X,Region,_), write('('),
    write(X), write(' '), write(Y), write(')'), nl, fail.
write_pixel_group3(Region).

/* Get the group type for a group and associate it with a letter. */
group_type(Group,'w',NewGlist) :- whale(_,Group,_), whale_group(WhaleGroup),
    member(Group,WhaleGroup), get_whalegroup_regions(WhaleGroup,[],NewGlist).
group_type(Group,'u',Glist) :- whale(_,Group,Glist).
group_type(Group,'u',Glist) :- unknown_group(_,Group,Glist).
group_type(Group,'q',Glist) :- possible_quake(_,Group,Glist).
group_type(Group,'o',Glist) :- group(Group,Glist).

/* Find all the regions that belong to one whale song and list them. */
get_whalegroup_regions([],WhaleList,WhaleList).
get_whalegroup_regions([Group|GroupList],RegionList,WhaleList) :-
    group(Group,Rlist),
    append(Rlist,RegionList,NewRegionList), assert(already_written(Group)),
    get_whalegroup_regions(GroupList,NewRegionList,WhaleList).

/* END END END END END END END END END END END END END END END */
/* SEEM SEEM SEEM SEEM SEEM SEEM SEEM SEEM SEEM SEEM SEEM SEEM SEEM SEEM SEEM */

/* Region "clumping" using the gradient-threshold array as input. */
/* This version more appropriate for sound processing. */

incrclump(Infile,Outfile) :- abolish(region_name_change/2),

```

```

abolish(lastrowegensym/1), abolish(final_code/2),
see(Infile), tell(tempclumpfile),
readlineclump(GL1), length(GL1,N), constant_vector(*,N,CV),
incrclumprow(CV,CV,GL1),
see(tempclumpfile), tell(Outfile),
abolish(lastrowegensym/1), fix_region_names, seen, told.
incrclumprow(RL1,GL1,GL2) :-
at_end_of_file,
incrclumprow2(RL1,*,[*|GL1],[*|GL2],RL2), write_line(RL2),
seen, told, !.
incrclumprow(RL1,GL1,GL2) :-
incrclumprow2(RL1,*,[*|GL1],[*|GL2],RL2),
write_line(RL2), readlineclump(GL3), incrclumprow(RL2,GL2,GL3).
/* Reads the first pass output back in, changes the region names, writes out */
fix_region_names :- not(at_end_of_file), readlineclump(L),
fix_region_names(L,NL), write_line(NL), !, fix_region_names.
fix_region_names :- abolish(region_name_change/2), abolish(final_code/2).
fix_region_names([],[]).
fix_region_names([*|L],[*|NL]) :- !, fix_region_names(L,NL), !.
fix_region_names([R1|L],[R2|NL]) :- transitive_region_name_change(R1,R3),
xfinal_code(R3,R2), !, fix_region_names(L,NL), !.
fix_region_names([R1|L],[R2|NL]) :- xfinal_code(R1,R2),
fix_region_names(L,NL), !.
xfinal_code(R1,R2) :- final_code(R1,R2), !.
xfinal_code(R1,R2) :- rowegensym(R2), assertz(final_code(R1,R2)), !.

/* Handles a single row during incremental clumping. */
/* First arg is region labels for previous row, second arg is region */
/* label for cell to the left of the current cell, third arg is list */
/* of 0s and *s (gradient threshold output) for row above, fourth arg */
/* is same for current row, and fifth arg is output, the new region labels. */

/* At end of row? */
incrclumprow2([],_,[_],[_],[]) :- !.
/* Is this an edge cell? */
incrclumprow2([R11|RL1],_,[_],G12|GL1,[G21,*|GL2],[*|RL2]) :- !,
incrclumprow2(RL1,*,[G12|GL1],[*|GL2],RL2).
/* Start a new region if this cell is unconnected above and to left */
incrclumprow2([R11|RL1],_,[_],*|GL1,[*,G22|GL2],[K|RL2]) :- !, rowegensym(K),
incrclumprow2(RL1,K,[*|GL1],[G22|GL2],RL2).
/* Merge with region above if this cell is unconnected to left */
incrclumprow2([R11|RL1],_,[_],G12|GL1,[*,G22|GL2],[R11|RL2]) :- !,
incrclumprow2(RL1,R11,[G12|GL1],[G22|GL2],RL2).
/* Merge with region left if this cell is unconnected above */
incrclumprow2([R11|RL1],R20,[_],*|GL1,[G21,G22|GL2],[R20|RL2]) :- !,
incrclumprow2(RL1,R20,[*|GL1],[G22|GL2],RL2).
/* Else merge two regions through this cell; lower region number wins */
incrclumprow2([R11|RL1],R20,[_],G12|GL1,[G21,G22|GL2],[RA|RL2]) :-
not(R20=R11), sort2(R20,R11,RA,RB), not(region_name_change(RB,RA)), !,
assertz(region_name_change(RB,RA)),
incrclumprow2(RL1,RA,[G12|GL1],[G22|GL2],RL2).
/* Avoid merge if two regions meeting are same or are marked for merge */
incrclumprow2([R11|RL1],R20,[_],G12|GL1,[G21,G22|GL2],[RA|RL2]) :- !,
sort2(R11,R20,RA,RB), incrclumprow2(RL1,RA,[G12|GL1],[G22|GL2],RL2).

/* Region "clumping" using the gradient-threshold array as input, */
/* with merging across diagonals also permitted. */
/* This version more appropriate for sound processing. */

diagincrclump(Infile,Outfile) :- abolish(region_name_change/2),
abolish(lastrowegensym/1), abolish(final_code/2),
see(Infile), tell(tempclumpfile),
readlineclump(GL1), asterisk_buffer(GL1,XGL1),
length(GL1,N), Np2 is N+2, constant_vector(*,Np2,CV),
diagincrclumprow(CV,CV,XGL1),
see(tempclumpfile), tell(Outfile),
abolish(lastrowegensym/1), fix_region_names, seen, told.
diagincrclumprow(XRL1,XGL1,XGL2) :- at_end_of_file,

```

```

diagincrclumprow2(XRL1,*,XGL1,XGL2,RL2), write_line(RL2), seen, told, !.
diagincrclumprow(XRL1,XGL1,XGL2) :-
diagincrclumprow2(XRL1,*,XGL1,XGL2,RL2),
write_line(RL2), readlineclump(GL3),
asterisk_buffer(RL2,XRL2), asterisk_buffer(GL3,XGL3),
diagincrclumprow(XRL2,XGL2,XGL3).
/* Puts asterisks at front and rear of list */
asterisk_buffer(L, [*|XL]) :- append(L, [*],XL), !.

/* Handles a single row during diagincremental clumping. */
/* First arg is region labels for previous row, second arg is region */
/* label for cell to the left of the current cell, third arg is list */
/* of 0s and *s (gradient threshold output) for row above, fourth arg */
/* is same for current row, and fifth arg is output, the new region labels. */

/* At end of row? */
diagincrclumprow2(_,_,[_,_],[_,_],[]) :- !.
/* Is this an edge cell? */
diagincrclumprow2([_|RL1],_,[_|GL1|GL2],[_|*|GL2],[*|RL2]) :- !,
diagincrclumprow2(RL1,*,[GL1|GL2],[*|GL2],RL2).
/* Start a new region if this cell is unconnected in all 4 directions */
diagincrclumprow2([*|*,*|RL1],*,[*|*|GL1],[*|*|GL2],[K|RL2]) :-
!, rowegensym(K), diagincrclumprow2([*|*|RL1],K,GL1,GL2,RL2).
/* Merge adjacent regions through this cell; lowest region number wins */
diagincrclumprow2([R11,R12,R13|RL1],R21,[_|GL1],[_|GL2],[K|RL2]) :-
sorted_numbers_in([R11,R12,R13,R21],[K|KL]),
update_neighbor_regions(K,KL),
diagincrclumprow2([R12,R13|RL1],K,GL1,GL2,RL2).
diagincrclumprow2(RL1,R21,GL1,GL2,RL2) :- told, seen,
write('Cannot clump this situation:'), nl,
write([RL1,R21,GL1,GL2,RL2]), nl, !, foo,
diagincrclumprow2(RL1,R21,GL1,GL2,RL2).

/* Returns the sorting of all numbers in a list of symbols */
sorted_numbers_in(L,SL) :- setof(N,number_member(N,L,NL), sort(NL,SL), !.
number_member(N,L) :- append(_,[N|_],L), number(N).
update_neighbor_regions(K,[]).
update_neighbor_regions(K,[K2|KL]) :-
(region_name_change(K2,K); assertz(region_name_change(K2,K))), !,
update_neighbor_regions(K,KL).

/* Function that returns the final region name to use after all merging */
transitive_region_name_change(R,R) :- not(region_name_change(R,R2)), !.
transitive_region_name_change(R1,R2) :- region_name_change(R1,R3), !,
transitive_region_name_change2(R3,R2).
transitive_region_name_change2(R1,R2) :- region_name_change(R1,R3),
transitive_region_name_change2(R3,R2).
transitive_region_name_change2(R,R) :- !.

/* Modified region "clumping" using the intensities themselves; */
/* compares differences of adjacent cells to grow_threshold. */
/* Puts -10000 boundary around the whole picture. */
/* This version more appropriate for picture brightness processing. */

incrxcclump(Infile,Outfile) :- abolish(region_name_change/2),
abolish(lastrowegensym/1), abolish(final_code/2),
see(Infile), tell(tempxcclumpfile),
readlineclump(GL1), length(GL1,N), constant_vector(-10000,N,CV),
incrxcclumprow(CV,CV,GL1),
see(tempxcclumpfile), tell(Outfile),
abolish(lastrowegensym/1), fix_region_names, seen, told.
incrxcclumprow(RL1,GL1,GL2) :-
at_end_of_file,
incrxcclumprow2(RL1,-10000,[-10000|GL1],[-10000|GL2],RL2), write_line(RL2),
seen, told, !.
incrxcclumprow(RL1,GL1,GL2) :-
incrxcclumprow2(RL1,-10000,[-10000|GL1],[-10000|GL2],RL2),
write_line(RL2), readlineclump(GL3), incrxcclumprow(RL2,GL2,GL3).

```

```

/* Handles a single row during incremental x-clumping. */
/* First arg is region labels for previous row, second arg is region */
/* label for cell to the left of the current cell, third arg is list */
/* of 0s and *s (gradient threshold output) for row above, fourth arg */
/* is same for current row, and fifth arg is output, the new region labels. */

/* At end of row? */
incrxcclumprow2([],_,[_],[_],[]) :- !.
/* Start a new region if this cell is unconnected above and to left */
incrxcclumprow2([R11|RL1],_,[_],G12|GL1,[G21,G22|GL2],[K|RL2]) :-
grow_threshold(G), deviation(G22,G12,DY), DY>G, deviation(G22,G21,DX), DX>G,
!, rowegensym(K), incrxcclumprow2(RL1,K,[G12|GL1],[G22|GL2],RL2).
/* Merge with region above if this cell is unconnected to left */
incrxcclumprow2([R11|RL1],_,[_],G12|GL1,[G21,G22|GL2],[R11|RL2]) :-
grow_threshold(G), deviation(G22,G21,DX), DX>G, !,
incrxcclumprow2(RL1,R11,[G12|GL1],[G22|GL2],RL2).
/* Merge with region left if this cell is unconnected above */
incrxcclumprow2([R11|RL1],R20,[_],G12|GL1,[G21,G22|GL2],[R20|RL2]) :-
grow_threshold(G), deviation(G22,G12,DY), DY>G, !,
incrxcclumprow2(RL1,R20,[G12|GL1],[G22|GL2],RL2).
/* Else merge two regions through this cell; lower region number wins */
incrxcclumprow2([R11|RL1],R20,[_],G12|GL1,[G21,G22|GL2],[RA|RL2]) :-
not(R20=R11), sort2(R20,R11,RA,RB), not(region_name_change(RB,RA)), !,
assertz(region_name_change(RB,RA)),
incrxcclumprow2(RL1,RA,[G12|GL1],[G22|GL2],RL2).
/* Avoid merge if two regions meeting are same or are marked for merge */
incrxcclumprow2([R11|RL1],R20,[_],G12|GL1,[G21,G22|GL2],[RA|RL2]) :- !,
sort2(R11,R20,RA,RB), incrxcclumprow2(RL1,RA,[G12|GL1],[G22|GL2],RL2).

/* Computes area, circumference, length and width of each region */
/* in an array of region codes. */

incrshapes(Infile,Outfile) :-
abolish(area/2), abolish(circumference/2), abolish(xmax/2),
abolish(xmin/2), abolish(ymax/2), abolish(ymin/2),
abolish(width/2), abolish(height/2),
see(Infile), readlineclump(L1), readlineclump(L2),
length(L1,N), constant_vector(*,N,CV),
incrshapes2(1,CV,L1,L2), seen,
tell(Outfile), listing(area), listing(circumference),
figure_dimensions, listing(width), listing(height), told.
incrshapes2(Row,L1,L2,L3) :- at_end_of_file, !,
length(L2,N), constant_vector(*,N,CV),
append(L1,[*],XL1), append(L2,[*],XL2), append(L3,[*],XL3),
incrshapes3(Row,1,[*|XL2],[*|XL1],[*|XL3]), Rowp1 is Row+1,
incrshapes3(Rowp1,1,[*|XL3],[*|XL2],[*|*|CV]), !.
incrshapes2(Row,L1,L2,L3) :- append(L1,[*],XL1), append(L2,[*],XL2),
append(L3,[*],XL3), incrshapes3(Row,1,[*|XL2],[*|XL1],[*|XL3]),
readlineclump(L4), Rowp1 is Row+1, incrshapes2(Rowp1,L2,L3,L4).
incrshapes3(Row,Col,[_],[_],[_],[_]).
incrshapes3(Row,Col,[I21,*|L2],[I11,I12|L1],[I31,I32|L3]) :- !,
Colp1 is Col+1, incrshapes3(Row,Colp1,[*|L2],[I12|L1],[I32|L3]).
incrshapes3(Row,Col,[I21,I22,I23|L2],[I11,I12|L1],[I31,I32|L3]) :-
uninteresting_region(I22), !, Colp1 is Col+1,
incrshapes3(Row,Colp1,[I22,I23|L2],[I12|L1],[I32|L3]).
incrshapes3(Row,Col,[I21,I22,I23|L2],[I11,I12|L1],[I31,I32|L3]) :- !,
update_area(I22), update_circumference(I22,I21,I12,I23,I32),
update_dimensions(I22,Row,Col), Colp1 is Col+1, !,
incrshapes3(Row,Colp1,[I22,I23|L2],[I12|L1],[I32|L3]).

update_area(R) :- retract(area(R,N)), !, Np1 is N+1, assertz(area(R,Np1)), !.
update_area(R) :- assertz(area(R,1)), !.

update_circumference(R,R,R,R,R) :- !.
update_circumference(R,_,_,_,_) :- retract(circumference(R,N)), !,
Np1 is N+1, assertz(circumference(R,Np1)), !.

```

```

update_circumference(R,_,_,_) :- assertz(circumference(R,1)), !.

update_dimensions(R,Row,Col) :-
    xmax(R,XH), Col=<XH, xmin(R,XL), Col>=XL,
    ymax(R,YH), Row=<YH, ymin(R,YL), Row>=YL, !.
update_dimensions(R,Row,Col) :-
    retract(xmax(R,XH)), retract(xmin(R,XL)),
    retract(ymax(R,YH)), retract(ymin(R,YL)), !,
    max(Col,XH,NXH), min(Col,XL,NXL), max(Row,YH,NYH), min(Row,YL,NYL),
    assertz(xmax(R,NXH)), assertz(xmin(R,NXL)),
    assertz(ymax(R,NYH)), assertz(ymin(R,NYL)), !.
update_dimensions(R,Row,Col) :-
    assertz(xmax(R,Col)), assertz(xmin(R,Col)),
    assertz(ymax(R,Row)), assertz(ymin(R,Row)), !.

figure_dimensions :- xmax(R,XH), xmin(R,XL), DX is 1+XH-XL,
    ymax(R,YH), ymin(R,YL), DY is 1+YH-YL,
    assertz(width(R,DX)), assertz(height(R,DY)), fail.
figure_dimensions.

/* Converts a picture file to pixel(X,Y,Value) facts, writes them out. */
/* All cells labelled "*" are ignored. */
pixel_convert(Infile,Outfile) :- abolish(pixel/3),
    see(Infile), pixel_convert2(1), seen,
    tell(Outfile), listing(pixel), told.
pixel_convert2(Row) :- not(at_end_of_file), readlineclump(L),
    pixel_convert3(1,Row,L), Rowpl is Row+1, !, pixel_convert2(Rowpl), !.
pixel_convert2(Row).
pixel_convert3(Col,Row,[I|L]) :- (I='*'; assertz(pixel(Col,Row,I))),
    Colpl is Col+1, !, pixel_convert3(Colpl,Row,L).
pixel_convert3(Col,Row,[]).

/* Reads a file and correlates its entries with current pixel(X,Y,Value) */
/* facts, makes the value at the corresponding place in the input file */
/* to be the fourth argument to a "pixel" which it writes out. */
pixel_join(Infile,Outfile) :-
    see(Infile), tell(Outfile), pixel_join2(1), seen, told.
pixel_join2(Row) :- not(at_end_of_file), readlineclump(L),
    pixel_join3(1,Row,L), Rowpl is Row+1, !, pixel_join2(Rowpl), !.
pixel_join2(Row).
pixel_join3(Col,Row,[I|L]) :- pixel(Col,Row,R), !,
    writeq(pixel(Col,Row,R,I)), write('.'), nl,
    Colpl is Col+1, !, pixel_join3(Colpl,Row,L).
pixel_join3(Col,Row,[I|L]) :- Colpl is Col+1, !, pixel_join3(Colpl,Row,L).
pixel_join3(Col,Row,[]).

/* Converts a picture file to CLIPS form and writes out a new file */
clips_convert(Infile,Outfile) :- see(Infile), tell(Outfile),
    write('(defacts incrvision)'), nl, clips_convert2(1),
    write(')'), seen, told.
clips_convert2(Row) :- not(at_end_of_file), readlineclump(L),
    clips_convert3(1,Row,L), Rowpl is Row+1, clips_convert2(Rowpl).
clips_convert2(Row).
clips_convert3(Col,Row,[*|L]) :- !, Colpl is Col+1, clips_convert3(Colpl,Row,L).
clips_convert3(Col,Row,[I|L]) :- write('(region_index)'), write(Col),
    write(' '), write(Row), write(' '), write(I), write(')'), nl,
    Colpl is Col+1, clips_convert3(Colpl,Row,L).
clips_convert3(Col,Row,[]).

/* Utility functions */

/* Some input routines. */
/* readlineclump(<list>)--reads a line from the terminal and */
/* clumps it into words, then binds its argument to this list */
/* readline(<list>)--reads a line from the terminal and */
/* binds its argument to this as a character string */
/* niceread(<list>)--reads a line from the terminal and */
/* binds its argument to its ascii list representation */

```

```

/* Note: This assumes that spaces, tabs, commas, colons, and */
/* semicolons separate words. If other symbols do too, */
/* add more "terminator" definitions. */

readlineclump(L) :- niceread(S), clumpstring(S,AL), makenames(AL,L), !.

makenames([],[]).
makenames([AL|LL],[NL|NLL]) :- name(NL,AL), makenames(LL,NLL).

clumpstring(L3,[L1|L5]) :- nextclump(L3,[],L1,L2), !, clumpstring(L2,L5).
clumpstring(L,[L]) :- member(X,L), not(terminator(X)), !.
clumpstring(L,[]).
nextclump([],L1,RL1,[]) :- not(L1=[]), !, reverse(L1,RL1).
nextclump([T|L],[],L2,L3) :- terminator(T), !, nextclump(L,[],L2,L3).
nextclump([T|L],L1,RL1,L) :- terminator(T), !, reverse(L1,RL1).
nextclump([X|L],L1,L2,L3) :- nextclump(L,[X|L1],L2,L3).
terminator(9).
terminator(32).
terminator(44).
terminator(58).
terminator(59).

readline(S) :- niceread(L), name(S,L).
niceread([]) :- at_end_of_file, !.
niceread([]) :- at_end_of_line, get0(_), !.
niceread([AC|AL]) :- get0(AC), niceread(AL).

constant_vector(Symbol,0,[]) :- !.
constant_vector(Symbol,Length,[Symbol|List]) :-
    Length1 is Length-1, constant_vector(Symbol,Length1,List), !.

sort2(N1,N2,N1,N2) :- N1<N2, !.
sort2(N1,N2,N2,N1) :- !.

/* Generates a new integer each time it is called */
rowegensym(K) :- retract(lastrowegensym(Km1)), !, K is Km1+1,
    asserta(lastrowegensym(K)).
rowegensym(1) :- asserta(lastrowegensym(1)).

```

## APPENDIX D: CLOS CODE FOR THE CORRELATION PROGRAM

```

(defconstant *region-match-criterion* 0.4)
;
(defmethod rb ()
  (require :process)
  (let ((blackboard1 (make-instance 'blackboard))
        (blackboard2 (make-instance 'blackboard))
        (picture (make-instance 'screen)))
    (initialize-blackboard blackboard1 'bone1e 'bone2e 0 3 'lower)
    (initialize-blackboard blackboard2 'bone3e 'bone4e 181 184 'upper)
    (start-picture picture)
    (run-blackboard-r blackboard1 picture)
    (run-blackboard-r blackboard2 picture)
    (match-far-phones blackboard1 blackboard2 picture)
  ))
;
(defmethod run-blackboard-r ((the-blackboard blackboard) (picture screen))
  (with-slots (phone1 phone2) the-blackboard
    (let ((new-region (gentemp)) listen-status)
      (set new-region (make-instance 'region))
      (if (< (phone-time phone1) (phone-time phone2))
          (let ()
            (setf listen-status (listen-to-hydrophone new-region phone1))
            (unless listen-status
              (setf listen-status (listen-to-hydrophone new-region phone2))))
          (let ()
            (setf listen-status (listen-to-hydrophone new-region phone2))
            (unless listen-status
              (setf listen-status (listen-to-hydrophone new-region phone1))))))
      (when listen-status
        (process-new-region new-region the-blackboard picture)
        (if (> (- (possible-region-match-time the-blackboard)
                  (last-definite-region-match-time the-blackboard))
            (* 6 (allowable-time-difference the-blackboard)))
            (match-definite-regions the-blackboard picture)
            (run-blackboard-r the-blackboard picture))))))
;
(defmethod initialize-blackboard ((the-blackboard blackboard) phone1-filename
                                   phone2-filename phone1-latitude phone2-latitude screen-position)
  (with-slots (phone1 phone2) the-blackboard
    (start-hydrophone phone1 phone1-filename phone1-latitude screen-position)
    (start-hydrophone phone2 phone2-filename phone2-latitude screen-position)
    (setf (screen-position the-blackboard) screen-position)
    (setf (allowable-time-difference the-blackboard)
          (* (/ (abs (- (latitude phone1) (latitude phone2))) 0.81) 1.5))))
  ;Speed of sound (SOS) through water = 0.81 nm/sec
  ;Slop factor for variation of SOS and phone separation is 1.5
;
(defmethod process-new-region
  (new-region (the-blackboard blackboard) (picture screen))
  (with-slots (phone1 phone2 unmatched-regions) the-blackboard
    (if (equal (phone-id (eval new-region)) (phone-id phone1))
        (setf (phone-time phone1) (time-min (eval new-region)))
        (setf (phone-time phone2) (time-min (eval new-region))))
    (change-unmatched-to-unmatchable new-region the-blackboard picture)
    (match-new-region-to-unmatched-regions new-region the-blackboard picture)
    (match-new-region-to-tentative-matched-regions new-region the-blackboard
      picture)
    (unless (member-p new-region (possible-matched-regions the-blackboard))
      (setf unmatched-regions (append unmatched-regions (list new-region)))
      (draw-region new-region the-blackboard picture 1))))
;

```



```

(defmethod change-unmatched-to-unmatchable
  (new-region (the-blackboard blackboard) (picture screen))
  (let ((unmatched-region-list (unmatched-regions the-blackboard)))
    (setf (unmatched-regions the-blackboard)
          (change-unmatched-to-unmatchable-r unmatched-region-list
        new-region the-blackboard picture))))
;
(defmethod change-unmatched-to-unmatchable-r (unmatched-region-list new-region
  (the-blackboard blackboard) (picture screen))
  (if unmatched-region-list
    (let ((tail (rest unmatched-region-list))
          (head (first unmatched-region-list)))
      (if (or (equal (phone-id (eval new-region)) (phone-id (eval head))))
        (<= (- (time-min (eval new-region)) (time-max (eval head)))
              (allowable-time-difference the-blackboard)))
        (cons head
              (change-unmatched-to-unmatchable-r tail new-region the-blackboard
            picture)))
      (let ()
        (setf (definite-unmatched-regions the-blackboard)
              (append (definite-unmatched-regions the-blackboard)
                    (list head))))
        (draw-region head the-blackboard picture 2)
        (change-unmatched-to-unmatchable-r tail new-region
      the-blackboard picture))))))
;
(defmethod match-new-region-to-unmatched-regions
  (new-region (the-blackboard blackboard) (picture screen))
  (let ((unmatched-region-list (unmatched-regions the-blackboard)))
    (setf (unmatched-regions the-blackboard)
          (match-new-region-to-unmatched-regions-r unmatched-region-list
        new-region the-blackboard picture))))
;
(defmethod match-new-region-to-unmatched-regions-r
  (unmatched-region-list new-region (the-blackboard blackboard) (picture screen))
  (if unmatched-region-list
    (let ((tail (rest unmatched-region-list))
          (head (first unmatched-region-list)))
      (if (equal (phone-id (eval new-region)) (phone-id (eval head)))
        (cons head (match-new-region-to-unmatched-regions-r tail new-region
      the-blackboard picture)))
        (let* ((region-match (match-region-factor new-region head)
              (allowable-time-difference the-blackboard))
              (region-matching-factor (first region-match)))
          (if (< region-matching-factor *region-match-criterion*)
            (cons head (match-new-region-to-unmatched-regions-r tail new-region
          the-blackboard picture)))
          (let ()
            (adjust-blackboard-for-match new-region head region-match
          the-blackboard picture)
            (match-new-region-to-unmatched-regions-r tail new-region
          the-blackboard picture))))))
;
(defmethod adjust-blackboard-for-match
  (new-region head region-match (the-blackboard blackboard) (picture screen))
  (let ((region-match-factor (first region-match))
        (match-dir (second region-match)))
    (unless (member (region-id (eval new-region))
                    (possible-matched-regions the-blackboard))
      (setf (possible-matched-regions the-blackboard)
            (append (possible-matched-regions the-blackboard) (list new-region)))
      (draw-region new-region the-blackboard picture 3))
    (setf (possible-matched-regions the-blackboard)
          (append (possible-matched-regions the-blackboard) (list head)))
    (let ((dir (direction new-region head match-dir (phone1 the-blackboard))))
      (setf (possible-region-matches the-blackboard)
            (append (possible-region-matches the-blackboard)
                  (list (list new-region head region-match-factor dir)))))

```

```

(draw-region head the-blackboard picture 3)
(connect-matched-region (list new-region head region-match-factor)
 picture (phone-id (eval new-region)))
(setf (possible-region-match-time the-blackboard
 (time-min (eval new-region)))))
;
(defmethod match-new-region-to-tentative-matched-regions
 (new-region (the-blackboard blackboard) (picture screen))
 (let ((matched-region-list (possible-matched-regions the-blackboard)))
 (setf (possible-matched-regions the-blackboard)
 (match-new-region-to-tentative-matched-regions-r matched-region-list
 new-region the-blackboard picture))))
;
(defmethod match-new-region-to-tentative-matched-regions-r
 (matched-region-list new-region (the-blackboard blackboard) (picture screen))
 (if matched-region-list
 (let ((tail (rest matched-region-list)) (head (first matched-region-list)))
 (if (or (equal (phone-id (eval new-region)) (phone-id (eval head)))
 (equal new-region head)
 (matched-member (list new-region head)
 (possible-region-matches the-blackboard))))
 (cons head (match-new-region-to-tentative-matched-regions-r tail new-region
 the-blackboard picture))
 (let* ((region-match (match-region-factor new-region head
 (allowable-time-difference the-blackboard)))
 (region-matching-factor (first region-match)))
 (if (< region-matching-factor *region-match-criterion*)
 (cons head (match-new-region-to-tentative-matched-regions-r tail
 new-region the-blackboard picture))
 (let ()
 (unless (member (region-id (eval new-region))
 (possible-matched-regions the-blackboard))
 (setf tail (append tail (list new-region))))
 (let ((dir (direction new-region head
 (second region-match) (phone1 the-blackboard))))
 (setf (possible-region-matches the-blackboard)
 (append (possible-region-matches the-blackboard)
 (list (list new-region head region-matching-factor
 dir))))))
 (draw-region new-region the-blackboard picture 3)
 (connect-matched-region
 (list new-region head region-matching-factor)
 picture (phone-id (eval new-region)))
 (setf (possible-region-match-time the-blackboard)
 (time-min (eval new-region)))
 (cons head (match-new-region-to-tentative-matched-regions-r
 tail new-region
 the-blackboard picture)))))))))
;
(defmethod match-region-factor (region1 region2 allowable-time-difference)
 (let* ((blob1 (eval region1)) (blob2 (eval region2)))
 (if (< allowable-time-difference
 (abs (- (time-min blob1) (time-min blob2))))
 '(0 0)
 (overlap-factor blob1 blob2))))
;
(defmethod matched-member (two-regions matched-list)
 (if matched-list
 (let ((head (first matched-list)) (tail (rest matched-list)))
 (if (and (equal (first two-regions) (first head))
 (equal (second two-regions) (second head)))
 t
 (matched-member two-regions tail))))))
;
(defmethod direction (region1 region2 offset (phone1 hydrophone))
 (let ((tmin1 (time-min (eval region1))) (tmin2 (time-min (eval region2))))
 (if (equal (phone-id (eval region1)) (phone-id phone1))
 (if (< tmin1 tmin2)

```

```

offset
(- offset))
(if (< tmin1 tmin2)
(- offset)
offset)))
;
(defmethod area-factor
((blob1 region) (blob2 region) (the-blackboard blackboard))
(let (factor)
(if (< (area blob1) (area blob2))
(setf factor (/ (area blob1) (area blob2)))
(setf factor (/ (area blob2) (area blob1))))
(let ((time-difference (abs (- (time-min blob1) (time-min blob2)))))
(if (> time-difference (* 2 (allowable-time-difference the-blackboard)))
(setf factor 0)
(setf factor (/ factor (- 1 (/ (* 0.1 time-difference)
(allowable-time-difference the-blackboard))))))
(if (> factor 1)
(setf factor 1)
factor)))
;
(defmethod overlap-factor
((blob1 region) (blob2 region))
(let ((overlap-value 0) overlap-index temp-overlap-value)
(do ((index (+ 2 (abs (- (time-min blob1) (time-min blob2))))) (1- index))
(= index (- (abs (- (time-min blob1) (time-min blob2))) 3))
overlap-value)
(if (< (time-min blob1) (time-min blob2))
(setf temp-overlap-value (overlap-region index blob1 blob2))
(setf temp-overlap-value (overlap-region index blob2 blob1)))
(when (> temp-overlap-value overlap-value)
(setf overlap-value temp-overlap-value)
(setf overlap-index index)))
(if overlap-index
(if (< (time-min blob1) (time-min blob2))
(list overlap-value overlap-index)
(list overlap-value (- overlap-index)))
'(0 0)))
;
(defmethod overlap-region (time-diff (blob1 region) (blob2 region))
(let (time-adjusted-region)
(setf time-adjusted-region (time-adjust time-diff (pixels blob1)))
(let* ((overlap-area (overlap-pixels time-adjusted-region (pixels blob2)))
(/ (* 2 overlap-area)
(+ (length (pixels blob1)) (length (pixels blob2))))))
;
(defmethod time-adjust (time-diff pixels)
(mapcar #'(lambda (x) (list (+ (first x) time-diff) (second x))) pixels))
;
(defmethod overlap-pixels (pixel-set1 pixel-set2)
; (if pixel-set1
; (if (pixel-member-p (first pixel-set1) pixel-set2)
; (append (list (first pixel-set1))
; (overlap-pixels (rest pixel-set1)
; (remove (first pixel-set1) pixel-set2)))
; (overlap-pixels (rest pixel-set1) pixel-set2)))
;
(defmethod overlap-pixels (region1 region2)
(let ((overlap-value 0))
(if (< (length region1) (length region2))
(dolist (pixel region1 overlap-value)
(if (pixel-member-p pixel region2)
(setf overlap-value (1+ overlap-value))))
(dolist (pixel region2 overlap-value)
(if (pixel-member-p pixel region1)
(setf overlap-value (1+ overlap-value))))))
;
(defmethod pixel-member-p (item-list list-of-item-lists)

```

```

(if list-of-item-lists
  (let ((head (first list-of-item-lists)) (tail (rest list-of-item-lists)))
    (if (and (= (first item-list) (first head))
              (= (second item-list) (second head)))
      t
      (pixel-member-p item-list tail))))))
;
(defmethod min-freq-factor ((blob1 region) (blob2 region))
  (let ((freq-factor
        (- 1 (/ (abs (- (freq-min blob2) (freq-min blob1))) 64))))
    (* freq-factor freq-factor)))
; There are 64 Hz in the frequency band.
;
(defmethod match-definite-regions ((the-blackboard blackboard) (picture screen))
  (let ((head (first (possible-region-matches the-blackboard)))
        (tail (rest (possible-region-matches the-blackboard))))
    (if head
      (match-definite-regions-r head tail the-blackboard picture)))
;
(defmethod match-definite-regions-r
  (possible-match possible-matched-list
   the-blackboard blackboard) (picture screen))
  (when possible-matched-list
    (if (check-time-frame-r possible-match the-blackboard)
      (let ((multiple-matches (find-multiple-matches-r possible-match
        possible-matched-list
        the-blackboard)))
        (if multiple-matches
          (perform-optimization-for-multiple-matches)
          (change-possible-match-to-definite possible-match
            the-blackboard picture))
        (match-definite-regions-r (first possible-matched-list)
          (rest possible-matched-list)
          the-blackboard picture))))))
;
(defmethod change-possible-match-to-definite
  (possible-match (the-blackboard blackboard) (picture screen))
  (setf (possible-region-matches the-blackboard)
        (remove possible-match (possible-region-matches the-blackboard)))
  (setf (definite-region-matches the-blackboard)
        (append (list possible-match) (definite-region-matches the-blackboard)))
  (setf (possible-matched-regions the-blackboard)
        (remove (first possible-match)
          (possible-matched-regions the-blackboard)))
  (setf (possible-matched-regions the-blackboard)
        (remove (second possible-match)
          (possible-matched-regions the-blackboard)))
  (setf (definite-matched-regions the-blackboard)
        (append (definite-matched-regions the-blackboard)
          (list (first possible-match))))
  (setf (definite-matched-regions the-blackboard)
        (append (definite-matched-regions the-blackboard)
          (list (second possible-match))))
  (setf (last-definite-region-match-time the-blackboard)
        (time-min (eval (first possible-match))))
  (draw-region (second possible-match) the-blackboard picture 4)
  (draw-region (first possible-match) the-blackboard picture 4))
;
(defmethod find-multiple-matches-r
  (matched-region region-matches (the-blackboard blackboard))
  (let ((head (first region-matches)))
    (if region-matches
      (if (region-in-common matched-region head)
        (when t
          (check-time-frame-r head the-blackboard)
          (cons head
            (find-multiple-matches-r matched-region (rest region-matches)
              the-blackboard))))
      ())))

```

```

(find-multiple-matches-r matched-region (rest region-matches)
the-blackboard)))
;
(defmethod region-in-common (matched-region1 matched-region2)
  (if (member (first matched-region1) matched-region2)
      (first matched-region1)
      (if (member (second matched-region1) matched-region2)
          (second matched-region1)
          (second matched-region2))))
;
(defmethod perform-optimization-for-multiple-matches ()
  (print "Perform-optimization-for-multiple-matches is a stub"))
;
(defmethod check-time-frame-r (possible-matched-region
the-blackboard blackboard)
  (if (< (time-max (eval (first possible-matched-region)))
      (- (possible-region-match-time the-blackboard)
         (* 3 (allowable-time-difference the-blackboard))))
      (if (< (time-max (eval (second possible-matched-region)))
          (- (possible-region-match-time the-blackboard)
             (* 3 (allowable-time-difference the-blackboard))))
          t)))

(defclass blackboard ()
  ((phone1 :accessor phone1
:iniform (make-instance 'hydrophone))
(phone2 :accessor phone2
:iniform (make-instance 'hydrophone))
(screen-position :accessor screen-position
:iniform 'lower)
(unmatched-regions :accessor unmatched-regions
:iniform nil)
(possible-matched-regions :accessor possible-matched-regions
:iniform nil)
(possible-region-matches :accessor possible-region-matches
:iniform nil)
(possible-region-match-time :accessor possible-region-match-time
:iniform 0)
(definite-matched-regions :accessor definite-matched-regions
:iniform nil)
(definite-region-matches :accessor definite-region-matches
:iniform nil)
(last-definite-region-match-time :accessor last-definite-region-match-time
:iniform 0)
(definite-unmatched-regions :accessor definite-unmatched-regions
:iniform nil)
(allowable-time-difference :accessor allowable-time-difference)))
(defclass blackboard-object () () )

(defclass hydrophone ()
  ((file :initarg :file
:accessor file)
(blackboard-id :initarg 'lower
:accessor blackboard-id)
(phone-id :initarg :phone-id
:accessor phone-id)
(latitude :initarg :latitude
:accessor latitude)
(phone-time :iniform 0
:accessor phone-time)))
;
(defclass region ()
  ((blackboard-id :initarg 'lower
:accessor blackboard-id)
(phone-id :initarg :phone-id
:accessor phone-id)
(region-id :initarg :region-id
:accessor region-id)

```

```

(region-type :accessor region-type)
(pixels :initarg :pixels
:accessor pixels)
(time-min :initarg :time-min
:accessor time-min)
(time-max :initarg :time-max
:accessor time-max)
(freq-min :initarg :freq-min
:accessor freq-min)
(freq-max :initarg :freq-max
:accessor freq-max)
(area :initarg :area
:accessor area)))
;
(defclass unmatched-regions (region)
  ((region-list :initform nil
:accessor region-list)))
;
(defmethod start-hydrophone ((phone hydrophone) filename input latitude
screen-position)
  (with-slots (file latitude phone-id blackboard-id) phone
    (setf file filename)
    (setf latitude input-latitude)
    (setf phone-id (gentemp))
    (setf blackboard-id screen-position)
    (setf file (open filename :direction :input))))
;
(defmethod listen-to-hydrophone (region-name (phone hydrophone))
  (let ((blob (eval region-name)))
    (setf (phone-id blob) (phone-id phone))
    (when (setf (pixels blob) (read (file phone) nil nil))
      (setf (blackboard-id blob) (blackboard-id phone))
      (setf (region-id blob) region-name)
      (setf (region-type blob) (first (pixels blob)))
      (setf (pixels blob) (rest (pixels blob)))
      (setf (time-min blob) (get-min-time (pixels blob)))
      (setf (time-max blob) (get-max-time (pixels blob)))
      (setf (freq-min blob) (get-min-freq (pixels blob)))
      (setf (freq-max blob) (get-max-freq (pixels blob)))
      (setf (area blob) (length (pixels blob))))))
;
(defmethod get-min-time (pixel-list)
  (get-min-time-r (first (first pixel-list)) (rest pixel-list)))
;
(defmethod get-min-time-r (temp-min pixel-list)
  (if (first pixel-list)
    (let ((next-time (first (first pixel-list))))
      (if (< temp-min next-time)
        (get-min-time-r temp-min (rest pixel-list))
        (get-min-time-r next-time (rest pixel-list)))
      temp-min))
;
(defmethod get-max-time (pixel-list)
  (get-max-time-r (first (first pixel-list)) (rest pixel-list)))
;
(defmethod get-max-time-r (temp-max pixel-list)
  (if (first pixel-list)
    (let ((next-time (first (first pixel-list))))
      (if (> temp-max next-time)
        (get-max-time-r temp-max (rest pixel-list))
        (get-max-time-r next-time (rest pixel-list)))
      temp-max))
;
(defmethod get-min-freq (pixel-list)
  (get-min-freq-r (second (first pixel-list)) (rest pixel-list)))
;
(defmethod get-min-freq-r (temp-min pixel-list)
  (if (first pixel-list)

```

```

(let ((next-freq (second (first pixel-list))))
  (if (< temp-min next-freq)
      (get-min-freq-r temp-min (rest pixel-list))
      (get-min-freq-r next-freq (rest pixel-list)))
  temp-min))
;
(defmethod get-max-freq (pixel-list)
  (get-max-freq-r (second (first pixel-list)) (rest pixel-list)))
;
(defmethod get-max-freq-r (temp-max pixel-list)
  (if (first pixel-list)
      (let ((next-freq (second (first pixel-list))))
        (if (> temp-max next-freq)
            (get-max-freq-r temp-max (rest pixel-list))
            (get-max-freq-r next-freq (rest pixel-list))))
      temp-max))

(defconstant *pixel-size* 1)
(defvar *offset* (* *pixel-size* 64))
(defconstant *low* 60)
(defconstant *high* 350)

(defclass screen ()
  ((id :accessor id)
   (borders :initform 5
    :accessor borders)
   (left :initform 0
    :accessor left)
   (bottom :initform 0
    :accessor bottom)
   (width :initform 1150
    :accessor width)
   (height :initform 700
    :accessor height)
   (title :initform "Hydrophone Region Matches"
    :accessor title)
   (activate-p :initform t
    :accessor activate-p)
   (lower-y-text-position :accessor lower-y-text-position
    :initform 1000)
   (upper-y-text-position :initform 1000
    :accessor upper-y-text-position)))

(defmethod start-picture ((picture screen))
  (require :xcw)
  (cw:initialize-common-windows)
  (setf (id picture) (cw:make-window-stream :borders (borders picture)
    :left (left picture)
    :bottom (bottom picture)
    :width (width picture)
    :height (height picture)
    :title (title picture)
    :activate-p (activate-p picture)))
  (setf (cw:window-stream-extent-width (id picture)) (* 2 (width picture)))
  (cw:enable-window-stream-extent-scrolling (id picture) :vertical nil
    :horizontal t)
  (draw-y-scale picture)
  (draw-time-line picture))
;
(defmethod draw-y-scale ((picture screen))
  (cw:draw-line-xy (id picture) 0 *low* (* 2 (width picture)) *low*)
  (cw:draw-line-xy (id picture) 0 (+ *low* (* *pixel-size* 64))
    (* 2 (width picture)) (+ *low* (* *pixel-size* 64)))
  (cw:draw-line-xy (id picture) 0 (+ *low* (* *pixel-size* 128))
    (* 2 (width picture)) (+ *low* (* *pixel-size* 128)))

```

```

(* 2 (width picture)) (+ *low* (* *pixel-size* 128)))
(cw:draw-line-xy (id picture) 0 *high* (* 2 (width picture)) *high*)
(cw:draw-line-xy (id picture) 0 (+ *high* (* *pixel-size* 64))
(* 2 (width picture)) (+ *high* (* *pixel-size* 64)))
(cw:draw-line-xy (id picture) 0 (+ *high* (* *pixel-size* 128))
(* 2 (width picture)) (+ *high* (* *pixel-size* 128)))
;
(defmethod draw-time-line ((picture screen))
  (do ((time-step 0 (+ (* *pixel-size* 60) time-step)))
    ((> time-step (* 2 (width picture))) nil)
    (cw:draw-line-xy (id picture) time-step *high* time-step (- *high* 3))
    (cw:draw-line-xy (id picture) time-step *low* time-step (- *low* 2))))
;
(defmethod paint-picture ((picture screen) (the-blackboard blackboard))
  (draw-regions-r (definite-unmatched-regions the-blackboard) the-blackboard
  picture 2)
  (draw-regions-r (possible-matched-regions the-blackboard) the-blackboard
  picture 3)
  (draw-regions-r (unmatched-regions the-blackboard) the-blackboard
  picture 1)
  (connect-matched-regions-r (possible-region-matches the-blackboard)
  picture (phone-id (phone1 the-blackboard))))
;
(defmethod draw-regions-r
  (region-list (the-blackboard blackboard) (picture screen) shade)
  (when region-list
    (draw-region (first region-list) the-blackboard picture shade)
    (draw-regions-r (rest region-list) the-blackboard picture shade)))
;
(defmethod draw-region
  (region (the-blackboard blackboard) (picture screen) shade)
  (with-slots (phone1 phone2) the-blackboard
    (let* ((blob (eval region)) (screen-position (blackboard-id (eval blob)))
    offset)
      (if (equal screen-position 'lower)
        (setf offset *low*)
        (setf offset *high*))
      (identify-region (id picture) region offset
        (phone-id blob) (phone-id phone2))
      (draw-pixels (id picture) (pixels blob) shade offset
        (phone-id blob) (phone-id phone2))))))
;
(defmethod draw-pixels
  (picture-id pixel-list shade offset phone-id phone2-id)
  (dolist (pixel pixel-list t)
    (let ((time (* *pixel-size* (first pixel)))
      (freq (* *pixel-size* (second pixel))))
      (if (equal phone-id phone2-id)
        (setf freq (+ *offset* freq))
        (setf freq (+ freq offset)))
      (if (= shade 1)
        (cw:draw-filled-rectangle-xy picture-id time freq
        *pixel-size* *pixel-size* :color cw:black)
        (if (= shade 2)
          (cw:draw-filled-rectangle-xy picture-id time freq
          *pixel-size* *pixel-size* :color cw:black)
          (if (= shade 3)
            (cw:draw-filled-rectangle-xy picture-id time freq
            *pixel-size* *pixel-size* :color cw:red)
            (if (= shade 4)
              (cw:draw-filled-rectangle-xy picture-id time freq
              *pixel-size* *pixel-size* :color cw:blue)
              (cw:draw-filled-rectangle-xy picture-id time freq
              *pixel-size* *pixel-size*))))))))))
;
(defmethod identify-region (picture-id region offset phone-id phone2-id)
  (let ((x (* *pixel-size* (time-max (eval region))))
    (y (* *pixel-size* (freq-max (eval region)))))

```



```

(if (equal (blackboard-id (eval region)) 'lower)
  (setf offset *low*)
  (setf offset *high*))
(setf y (+ y offset))
(if (equal phone-id phone2-id)
  (setf y (+ y *offset*)))
(setf (cw:window-stream-x-position picture-id) x)
(setf (cw:window-stream-y-position picture-id) (+ y 4))
(format picture-id "~a" (region-type (eval region))))
;
(defmethod connect-matched-regions-r (region-matches (picture screen) phone1-id)
  (when region-matches
    (connect-matched-region (first region-matches) picture phone1-id)
    (connect-matched-regions-r (rest region-matches) picture phone1-id)))
;
(defmethod connect-matched-region (region-match (picture screen) phone1-id)
  (let* ((region1 (first region-match)) (region2 (second region-match))
        (pixel1 (middle (pixels (eval region1))))
        (pixel2 (middle (pixels (eval region2)))))
    (x1 (* *pixel-size* (first pixel1)))
    (x2 (* *pixel-size* (first pixel2)))
    (y1 (* *pixel-size* (second pixel1)))
    (y2 (* *pixel-size* (second pixel2))) offset)
    (if (equal (blackboard-id (eval region1)) 'lower)
      (setf offset *low*)
      (setf offset *high*))
    (setf y1 (+ y1 offset))
    (setf y2 (+ y2 offset))
    (if (equal (phone-id (eval region1)) phone1-id)
      (cw:draw-line-xy (id picture) x1 y1 x2 (+ *offset* y2))
      (cw:draw-line-xy (id picture) x1 (+ *offset* y1) x2 y2))
    (if (equal (phone-id (eval region1)) phone1-id)
      (setf (cw:window-stream-x-position (id picture)) x2)
      (setf (cw:window-stream-x-position (id picture)) x1))
    (setf (cw:window-stream-y-position (id picture))
      (set-y-text-position picture region1))
    (format (id picture) "~d" (time-min (eval region1))))
  ; (format (id picture) "~3,2f" (third region-match)))
;
(defmethod set-y-text-position ((picture screen) region-name)
  (if (equal (blackboard-id (eval region-name)) 'lower)
    (if (> (lower-y-text-position picture) 40)
      (setf (lower-y-text-position picture) 2)
      (setf (lower-y-text-position picture)
        (+ (lower-y-text-position picture) 15)))
    (if (> (upper-y-text-position picture)
      (+ *high* (+ 55 (* *pixel-size* 128))))
      (setf (upper-y-text-position picture) (+ *high* 15 (* *pixel-size* 128)))
      (setf (upper-y-text-position picture)
        (+ (upper-y-text-position picture) 15))))
;
(defmethod draw-match-list (matched-list
  (blackboard1 blackboard) (blackboard2 blackboard) (picture screen))
  (dolist (matched-list-1 matched-list nil)
    (let ((matched-pair-1 (first matched-list-1)) pixel1 x1 y1)
      (if (equal (blackboard-id (eval (first matched-pair-1))) 'lower)
        (if (equal (screen-position blackboard1) 'lower)
          (if (equal (phone-id (eval (first matched-pair-1)))
            (phone-id (phone1 blackboard1)))
            (let ()
              (setf pixel1 (middle (pixels (eval (second matched-pair-1)))))
              (setf x1 (* *pixel-size* (first pixel1)))
              (setf y1 (+ (* *pixel-size* (second pixel1)) *low* *offset*)))
              (let ()
                (setf pixel1 (middle (pixels (eval (first matched-pair-1)))))
                (setf x1 (* *pixel-size* (first pixel1)))

```

```

(setf y1 (+ (* *pixel-size* (second pixel1)) *low* *offset*)))
(if (equal (phone-id (eval (first matched-pair-1)))
(phone-id (phonel blackboard2)))
(let ()
(setf pixel1 (middle (pixels (eval (second matched-pair-1)))))
(setf x1 (* *pixel-size* (first pixel1)))
(setf y1 (+ (* *pixel-size* (second pixel1)) *low* *offset*)))
(let ()
(setf pixel1 (middle (pixels (eval (first matched-pair-1)))))
(setf x1 (* *pixel-size* (first pixel1)))
(setf y1 (+ (* *pixel-size* (second pixel1)) *low* *offset*))))
(let ()
(if (equal (phone-id (eval (first matched-pair-1)))
(phone-id (phonel blackboard1)))
(let ()
(setf pixel1 (middle (pixels (eval (first matched-pair-1)))))
(setf x1 (* *pixel-size* (first pixel1)))
(setf y1 (+ (* *pixel-size* (second pixel1)) *high*)))
(let ()
(setf pixel1 (middle (pixels (eval (second matched-pair-1)))))
(setf x1 (* *pixel-size* (first pixel1)))
(setf y1 (+ (* *pixel-size* (second pixel1)) *high*)))
(if (equal (phone-id (eval (first matched-pair-1)))
(phone-id (phonel blackboard2)))
(let ()
(setf pixel1 (middle (pixels (eval (first matched-pair-1)))))
(setf x1 (* *pixel-size* (first pixel1)))
(setf y1 (+ (* *pixel-size* (second pixel1)) *high*)))
(let ()
(setf pixel1 (middle (pixels (eval (second matched-pair-1)))))
(setf x1 (* *pixel-size* (first pixel1)))
(setf y1 (+ (* *pixel-size* (second pixel1)) *high*))))
(dolist (matched-list-2 (rest matched-list-1) nil)
(dolist (matched-pair-2 matched-list-2 nil)
; (let ((matched-pair-2 (first matched-list-2)))
(when (> (first (first matched-pair-2)) 0.5)
(let* ((pixel2 (middle
(pixels (eval (first (second matched-pair-2)))))
(x2 (* *pixel-size* (first pixel2)))
(y2 (* *pixel-size* (second pixel2))))
(if (equal (blackboard-id (eval (first (second matched-pair-2))))
'lower)
(setf y2 (+ y2 *low*))
(setf y2 (+ y2 *high*)))
(if (equal (blackboard-id (eval (first (second matched-pair-2))))
'lower)
(setf y2 (+ y2 *offset*)))
(cw:draw-line-xy (id picture) x1 y1 x2 y2))))))
;
(defmethod draw-best-match-list (matched-list
(blackboard1 blackboard) (blackboard2 blackboard) (picture screen))
(dolist (matched-list-1 matched-list nil)
(let ((matched-pair-1 (first matched-list-1)) pixel1 x1 y1)
(if (equal (blackboard-id (eval (first matched-pair-1))) 'lower)
(if (equal (screen-position blackboard1) 'lower)
(if (equal (phone-id (eval (first matched-pair-1)))
(phone-id (phonel blackboard1)))
(let ()
(setf pixel1 (middle (pixels (eval (second matched-pair-1)))))
(setf x1 (* *pixel-size* (first pixel1)))
(setf y1 (+ (* *pixel-size* (second pixel1)) *low* *offset*)))
(let ()
(setf pixel1 (middle (pixels (eval (first matched-pair-1)))))
(setf x1 (* *pixel-size* (first pixel1)))
(setf y1 (+ (* *pixel-size* (second pixel1)) *low* *offset*))))
(if (equal (phone-id (eval (first matched-pair-1)))
(phone-id (phonel blackboard2)))
(let ()

```

```

(setf pixel1 (middle (pixels (eval (second matched-pair-1))))
(setf x1 (* *pixel-size* (first pixel1)))
(setf y1 (+ (* *pixel-size* (second pixel1)) *low* *offset*)))
(let ()
  (setf pixel1 (middle (pixels (eval (first matched-pair-1))))
  (setf x1 (* *pixel-size* (first pixel1)))
  (setf y1 (+ (* *pixel-size* (second pixel1)) *low* *offset*))))
(let ()
  (if (equal (phone-id (eval (first matched-pair-1)))
  (phone-id (phone1 blackboard1)))
  (let ()
    (setf pixel1 (middle (pixels (eval (first matched-pair-1))))
    (setf x1 (* *pixel-size* (first pixel1)))
    (setf y1 (+ (* *pixel-size* (second pixel1)) *high*))))
  (setf pixel1 (middle (pixels (eval (second matched-pair-1))))
  (setf x1 (* *pixel-size* (first pixel1)))
  (setf y1 (+ (* *pixel-size* (second pixel1)) *high*))))
  (if (equal (phone-id (eval (first matched-pair-1)))
  (phone-id (phone1 blackboard2)))
  (let ()
    (setf pixel1 (middle (pixels (eval (first matched-pair-1))))
    (setf x1 (* *pixel-size* (first pixel1)))
    (setf y1 (+ (* *pixel-size* (second pixel1)) *high*))))
  (setf pixel1 (middle (pixels (eval (second matched-pair-1))))
  (setf x1 (* *pixel-size* (first pixel1)))
  (setf y1 (+ (* *pixel-size* (second pixel1)) *high*))))
  (let ()
    (setf pixel1 (middle (pixels (eval (second matched-pair-1))))
    (setf x1 (* *pixel-size* (first pixel1)))
    (setf y1 (+ (* *pixel-size* (second pixel1)) *high*))))
  ;(dolist (matched-list-2 (rest matched-list-1) nil)
  ;(dolist (matched-pair-2 matched-list-2 nil)
  (let ((matched-pair-2 (second matched-list-1)))
  ;(when (> (first (first matched-pair-2)) 0.5)
  (let* ((pixel2 (middle
  (pixels (eval (first matched-pair-2))))
  (x2 (* *pixel-size* (first pixel2)))
  (y2 (* *pixel-size* (second pixel2))))
  (if (equal (blackboard-id (eval (first matched-pair-2)))
  'lower)
  (setf y2 (+ y2 *low*))
  (setf y2 (+ y2 *high*)))
  (if (equal (blackboard-id (eval (first matched-pair-2)))
  'lower)
  (setf y2 (+ y2 *offset*)))
  (cw:draw-line-xy (id picture) x1 y1 x2 y2))))))

```

```

(defconstant SLICES 20)
(defconstant *far-threshold* 0.15)

```

```

(defmethod match-far-phones
  ((blackboard1 blackboard) (blackboard2 blackboard) (picture screen))
  (let ((match-list1 nil) (match-list2 nil) (best-match-list nil)
  (hough-list nil) (new-match-list1 nil) (new-match-list2 nil)
  (allowable-time-difference
  (* (/ (abs (- (latitude (phone1 blackboard1))
  (latitude (phone2 blackboard2)))) 0.81) 1.2)))
  (dolist (matched-pair (definite-region-matches blackboard1) nil)
  (setf match-list1
  (append (list (list matched-pair
  (match-far-regions allowable-time-difference
  matched-pair blackboard2)))
  match-list1)))
  (dolist (matched-pair (definite-region-matches blackboard2) nil)
  (setf match-list2
  (append (list (list matched-pair
  (match-far-regions allowable-time-difference

```

```

matched-pair blackboard1)))
match-list2)))
(print "MATCH-LIST1") (print match-list1)
(print "MATCH-LIST2") (print match-list2)
(setf hough-list (get-hough-transform allowable-time-difference
match-list1 match-list2))
(setf new-match-list1 (apply-hough-list allowable-time-difference 1
hough-list match-list1))
; (print "MATCH-LIST1 after HOUGH") (print new-match-list1)
(setf new-match-list2 (apply-hough-list allowable-time-difference 1
hough-list match-list2))
; (print "MATCH-LIST2 after HOUGH") (print new-match-list2)
(setf best-match-list (get-best-matches new-match-list1 new-match-list2))
(draw-best-match-list best-match-list blackboard1 blackboard2 picture)
(dolist (matched-pair best-match-list nil)
(setf match-list1 (remove-main-match (first matched-pair) match-list1))
(setf match-list2 (remove-main-match (second matched-pair) match-list2))
(dolist (matched-pair best-match-list nil)
(setf match-list2 (remove-matched-pair (first matched-pair) match-list2))
(setf match-list1 (remove-matched-pair (second matched-pair) match-list1))
(print "MATCH-LIST1 AFTER REMOVAL") (print match-list1)
(print "MATCH-LIST2 AFTER REMOVAL") (print match-list2)
(setf hough-list (get-hough-transform allowable-time-difference
match-list1 match-list2))
(setf new-match-list1 (apply-hough-list allowable-time-difference 1
hough-list match-list1))
(setf new-match-list2 (apply-hough-list allowable-time-difference 1
hough-list match-list2))
(setf best-match-list (append best-match-list
(get-best-matches new-match-list1 new-match-list2)))
(draw-best-match-list best-match-list blackboard1 blackboard2 picture)
(print "BEST-MATCH-LIST") (print best-match-list)
(dolist (matched-pair best-match-list nil)
(setf match-list1 (remove-main-match (first matched-pair) match-list1))
(setf match-list2 (remove-main-match (second matched-pair) match-list2))
(dolist (matched-pair best-match-list nil)
(setf match-list2 (remove-matched-pair (first matched-pair) match-list2))
(setf match-list1 (remove-matched-pair (second matched-pair) match-list1))
(print "MATCH-LIST1 AFTER SECOND REMOVAL") (print match-list1)
(print "MATCH-LIST2 AFTER SECOND REMOVAL") (print match-list2)
))
; (draw-match-list match-list1 blackboard1 blackboard2 picture)))
;
(defmethod match-far-regions
(allowable-time-difference matched-pair-1 (the-blackboard blackboard))
(let ((region-1 (if (> (area (eval (first matched-pair-1)))
(area (eval (second matched-pair-1))))
(first matched-pair-1)
(second matched-pair-1)))
(list-of-matches nil))
(dolist (matched-pair-2 (definite-region-matches the-blackboard) nil)
(let* ((region-2 (if (> (area (eval (first matched-pair-2)))
(area (eval (second matched-pair-2))))
(first matched-pair-2)
(second matched-pair-2)))
(match-factor (match-region-factor
region-1 region-2 allowable-time-difference)))
(if (> (first match-factor) 0)
(setf list-of-matches
(append (list (list match-factor matched-pair-2))
list-of-matches))))
(if list-of-matches
(bubble-sort list-of-matches))))
;
; The Common Lisp Companion pg. 378
(defmethod bubble-sort (input-list)
(let (done)
(loop

```



```

(nth index hough-list))))))
(if (third (second best-match-list))
  (if (> (first (first (third (second best-match-list)))) 0.5)
    (let ((index (floor (+ offset (* -1.0
      (/ (second (first (third (second best-match-list))))
      time-slice))))))
      (if (< index 0) (setf index 0))
      (setf (nth index hough-list)
        (+ (first (first (third (second best-match-list))))
        (nth index hough-list))))))
    (dolist (element hough-list nil)
      (if (< element minimum) (setf minimum element))
      (setf hough-list (mapcar #'(lambda (x) (- x minimum)) hough-list))
      (setf hough-list (mapcar #'(lambda (x) (sqrt x)) hough-list))
      (dolist (element hough-list nil)
        (if (> element maximum) (setf maximum element))
        (if (> maximum 0.0)
          (setf hough-list (mapcar #'(lambda (x) (double-float
            (/ x maximum))) hough-list))
          (print "HOUGH-LIST")
          (print hough-list)
          hough-list)))
      )
    (defmethod apply-hough-list
      (allowable-time-difference direction hough-list match-list)
      (let ((time-slice (* (/ allowable-time-difference SLICES) 2.0))
        (offset (/ SLICES 2.0)) (new-match-list nil))
        (dolist (sub-match-list match-list new-match-list)
          (setf new-match-list
            (append new-match-list
              (list (list (first sub-match-list)
                (apply-hough-list2 (rest sub-match-list)
                  direction time-slice offset hough-list))))))
          )
        (defmethod apply-hough-list2
          (sub-sub-match-list direction time-slice offset hough-list)
          (let ((new-list nil))
            (dolist (match-element (first sub-sub-match-list) new-list)
              (setf new-list
                (append new-list (list (list (apply-hough-list3 (first match-element)
                  direction time-slice offset hough-list)
                  (second match-element))))))
              (if new-list (setf new-list (bubble-sort new-list))))
              )
            (defmethod apply-hough-list3
              (list-element direction time-slice offset hough-list)
              (let ((index (floor (+ offset (* direction
                (/ (second list-element) time-slice))))))
                (if (>= index SLICES) (setf index (1- SLICES)))
                (if (< index 0) (setf index 0))
                (list (* (first list-element) (nth index hough-list))
                  (second list-element))))
                )
            (defmethod get-best-matches (match-list1 match-list2)
              (let ((best-match-list nil) match-list-1 match-list-2)
                (setf match-list-1 match-list1)
                (setf match-list-2 match-list2)
                (dolist (matched-pair-1 match-list-1 nil)
                  (when (match-equal-p (first matched-pair-1)
                    (get-best-match (second (first (second matched-pair-1)))
                    match-list-2))
                    (print "best value =")
                    (print (float (first (first (first (second matched-pair-1))))))
                    (if (> (first (first (first (second matched-pair-1))))
                      *far-threshold*)
                      (setf best-match-list
                        (append (list (list (first matched-pair-1)
                          (second (first (second matched-pair-1))))
                          )
                        )
                    )
                    )
                )
              )

```

```

best-match-list))))))
best-match-list))
;
(defmethod match-equal-p (matched-pair-1 matched-pair-2)
  (if (equal (first matched-pair-1) (first matched-pair-2))
      t
      nil))
;
(defmethod get-best-match (matched-pair-1 match-list)
  (when match-list
    (if (match-equal-p matched-pair-1 (first (first match-list)))
        (let ()
          (when (second (first match-list))
            (second (first (second (first match-list))))))
        (get-best-match matched-pair-1 (rest match-list)))))
;
(defmethod remove-matched-pair (target-matched-pair match-list)
  (if match-list
      (append (list (remove-matched-pair-sub1
                    target-matched-pair (first match-list)))
              (remove-matched-pair target-matched-pair (rest match-list))))
      nil)
;
(defmethod remove-matched-pair-sub1 (target-matched-pair sub-list1)
  (append (list (first sub-list1))
          (list (remove-matched-pair-sub2 target-matched-pair (second sub-list1)))))
;
(defmethod remove-matched-pair-sub2 (target-matched-pair sub-list2)
  (if (first sub-list2)
      (if (match-equal-p target-matched-pair (second (first sub-list2)))
          (rest sub-list2)
          (append (list (first sub-list2))
                  (remove-matched-pair-sub2 target-matched-pair (rest sub-list2)))))
      nil)
;
(defmethod remove-main-match (matched-pair match-list)
  (if match-list
      (if (match-equal-p matched-pair (first (first match-list)))
          (rest match-list)
          (append (list (first match-list))
                  (remove-main-match matched-pair (rest match-list)))))
      nil)

```

## LIST OF REFERENCES

- Ballard, Dana H., and Brown, Christopher M., Computer Vision, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1982.
- Charniak, E., and McDermott, D., Introduction to Artificial Intelligence, Addison-Wesley, Reading, MA, 1985.
- Cummings, William C., and Thompson, Paul O., "Underwater Sounds from the Blue Whale, *Balaenoptera musculus*," The Journal of the Acoustical Society of America, v. 50, No. 4 (Part 2), 1971, pp. 1193 - 1198.
- Cummings, William C., Thompson, Paul O., and Ha, Samuel J., "*SOUNDS FROM BRYDE, BALAENOPTERA EDENI, AND FINBACK, B. PHYSALUS, WHALES IN THE GULF OF CALIFORNIA*," Fishery Bulletin, v. 84, No. 2, 1986, pp. 359 - 370.
- Dziak, Robert, Personal Communication, 1993.
- Fox, Christopher G., "NOAA's Use of U. S. Navy Fixed Hydrophone Arrays for Monitoring the Juan de Fuca Ridge," Poster Presentation at the Fall 1992 Meeting of the American Geophysical Union.
- Fox, Christopher G., "A Navy/NOAA Dual-Use Program in Ocean Monitoring: The Concept and its Feasibility", Unpublished Manuscript, 1993.
- Fox, Christopher G., Personal Communication, 1993.
- Hammond, Stephen R., and Walker, Daniel A., "Ridge Event Detection: T-Phase Signals from the Juan de Fuca Spreading Center", Marine Geophysical Researches, v. 13, 1991, pp 331-348.
- Loje, Kenneth F., Classes and Characteristics of Models. In: Modeling of Sound Propagation in the Ocean, Course Notes. Catholic Universtiy, 1983
- McDonald, Mark A., Personal Communication, 1993.
- Mizroch, S. A., Rice, Dale W., and Breiwick, Jeffrey M., "The Blue Whale, *Balaenoptera musculus*," Marine Fisheries Reviews, v. 46, No. 4, 1984, pp 15 - 19.
- Mizroch, S. A., Rice, Dale W., and Breiwick, Jeffrey M., "The Fin Whale, *Balaenoptera physalus*," Marine Fisheries Reviews, v. 46, No. 4, 1984, pp 20 - 24.



Nii, H. Penny, "Blackboard Systems Blackboard: Blackboard Application Systems, Blackboard Systems from a Knowledge Engineering Perspective (Part 2)", The AI Magazine, v. 7, No. 3, 1986, pp 82 - 106.

Press, William H., Flannery, Brian P., Teukolsky Saul A., and Vetterling, William T., Numerical Recipes in C, Cambridge University Press, New York, New York, 1988.

Rowe, Neil C., Artificial Intelligence Through Prolog, Prentice Hall, Englewood Cliffs, New Jersey, 1988.

Urick, Robert J., Principles of Underwater Sound, McGraw-Hill Book Company, New York, 1976

## INITIAL DISTRIBUTION LIST

- |    |  |   |
|----|--|---|
| 1. | Defense Technical Information Center<br>Cameron Station<br>Alexandria, VA 22304-6145   | 2 |
| 2. | Dudley Knox Library<br>Code 52<br>Naval Postgraduate School<br>Monterey, CA 93943-5002   | 2 |
| 3. | Dr. Ted Lewis<br>Code 37, Computer Science Department<br>Naval Postgraduate School<br>Monterey, CA 93943-5118  | 1 |
| 4. | Dr. Neil C. Rowe<br>Code CS/Rp<br>Associate Professor, Computer Science Department<br>Naval Postgraduate School<br>Monterey, CA 93943-5118                   | 3 |
| 5. | Dr. Christopher C. Fox<br>Adjunct Professor, Oregon State University<br>2115 S. E. OSU Dr.<br>Newport, OR 97365  | 2 |
| 6. | Dr. Monique Fargues<br>Code EC/Fa<br>Assistant Professor, Electronics and Computer Engineering Dept.<br>Naval Postgraduate School<br>Monterey, CA 93943-5121 | 1 |
| 7. | LCDR Dennis A. Seem<br>NOAA Ship Miller Freeman<br>Pacific Marine Center<br>1801 Fairview Ave., E.<br>Seattle, WA 98102                                      | 3 |